Universidad Autónoma de Madrid
Escuela politécnica superior

Double Degree in Computer Engineering and Mathematics

B. Sc. Thesis

IMAGE SYNTHESIS THROUGH FEATURE-BASED MIXING OF CONTENTS

Author: Daniel Villar Lora
Tutor: Simone Santini

JUN 2018

Universidad Autónoma de Madrid
Escuela politécnica superior

Doble Grado en Ingeniería Informática y Matemáticas

Trabajo de Fin de Grado

SÍNTESIS DE IMÁGENES MEDIANTE MEZCLA DE CONTENIDOS BASADOS EN CARACTERÍSTICAS

Author: Daniel Villar Lora
Tutor: Simone Santini

JUN 2018

IMAGE SYNTHESIS THROUGH FEATURE-BASED MIXING OF CONTENTS

Author: Daniel Villar Lora
Tutor: Simone Santini

Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
JUN 2018

Resumen

Resumen

En este proyecto abordamos el problema de sintetizar una imagen combinando varias características de dos imágenes de muestra. Una de estas características es el contenido, que hace referencia a las formas y la distribución espacial de los elementos presentes en la imagen. El otro es el estilo, que incluye colores, pinceladas, iluminación... Este problema se conoce como el problema del 'estilo neuronal'.

Comenzamos con una descripción general del estado del arte para analizar algunos de los métodos y herramientas más populares para implementar una solución a este problema: aprendizaje automático, redes convolucionales, descenso de gradiente... Además, describimos los métodos y herramientas elegidos en nuestra implementación. En particular, el objeto que hará la extracción de características será una red convolucional específica llamada VGG16.

A continuación, explicamos toda la estructura matemática detrás de la idea de cómo la red convolucional puede actuar como extractor. En primer lugar, analizamos el problema general de la síntesis de imágenes de manera abstracta. En segundo lugar, se usa un ejemplo concreto para mostrar cómo podemos implementar el método. Finalmente, ofrecemos una implementación del método de transferencia de estilo neuronal utilizando el enfoque de aprendizaje automático.

Por último, el resultado de nuestra implementación se usa para estudiar cómo funciona la red convolucional. Para hacerlo, hemos diseñado algunos experimentos, divididos en tres categorías. En el primero, analizamos el interior de la red, obteniendo algunas ideas sobre su arquitectura y lo que hace exactamente. En el segundo, tratamos de usar la red combinada con nuestro programa para generar un efecto de morphing. Los resultados de este experimento no fueron muy buenos, pero dejan mucho espacio para mejorar. En la tercera parte, estudiamos qué sucede cuando modificamos algunas de las características de la implementación.

En resumen, los resultados obtenidos de los experimentos son lo suficientemente satisfactorios: nos permiten cumplir nuestros objetivos para el proyecto y, a la vez, dar lugar a mejoras para el trabajo futuro.

Palabras Clave

Estilo neuronal, red convolucional, descenso por gradiente, caracterśticas de contenido y estilo, propagación hacia atrás, aprendizaje automático, VGG16, arte

Abstract

In this project we approach the problem of synthesizing an image by combining several features from two sample images. One of this features is the content, which makes reference to shapes and spatial distribution of the elements present in the image. The other is the style, which includes colors, brush strokes, illumination... This problem is known as the 'neural style' problem.

We start with an overview of the state of art to analyze some of the most popular methods and tools to implement a solution for this problem: machine learning, convolutional networks, gradient descent... Also, we describe the chosen methods and tools that we will use in our implementation. In particular, the object that will do the feature extraction will be a specific convolutional network called VGG16.

Afterwards, we explain all the entire mathematical structure behind the idea of how the convolutional network can act as the extractor. Firstly, we analyze the general problem of image synthesis abstractly. Secondly, a concrete example is used to show how we can implement the method. Finally, we provide an implementation of the neural style transfer method using the machine learning approach.

Lastly, the output of our implementation is used to study how the convolutional network works. In order to do it, we have designed some experiments, divided in three categories. In the first one, we analize the inside of the network, getting some insights about its architechture and what it does exactly. In the second one, we try to use the network combined with our program to generate a morphing effect. The results of this experiment did not come up quite good, but they leave plenty of room open for improvement. In the third category, we study what happens when we modify some of the characteristics of the implementation.

To sum up, the results obtained from the experiments are satisfying enough: they allow us to meet our objectives for the project while giving some room for improvements in future work.

Key words

Neural style, convolutional network, gradient descent, content and style features, backpropagation, machine learning, VGG16, art

Acknowledgments

To my parents and sister, who made this journey possible.
To my friends, who made this journey easier.
To Simone and Margarita, the best tutors of this university.
Thank you all.

Contents

Figures Index

Table index

1 Introduction

1.1 Motivation of the project

Visual communication is the conveyance of ideas and information in forms that can be seen. The use of images instead of text is around 15.4 faster: on the one hand, MIT neuroscientists have found that we can process an image in as little as 13 milliseconds [1]. On the other hand, Cambridge neuroscientists have found that we can integrate the different processes that lead to word recognition in 200 milliseconds [2]. Visual information also has an effect in our emotions and perception of reality, because our brain uses images to think [3] so certain feelings and lines of thought can be triggered after processing certain images. This two characteristics, that is, the cognitive and emotional aspects of vision are what makes it powerful: visual communication is faster, clearer and it doesn't depend on the viewer's language.

As a result, the ability to alter and manipulate visual information becomes interesting, because we can create new symbols using existing ones. Some examples of this can be found in paintings, photographies or films, like the fade-out effect to symbolize the dissapearance of something, someone or the passage of a sizable amount of time, certain poses or colours to depict victory or success... Some techniques include enhancing the properties of the image such as filtering noise or increasing the contrast like Instagram filters. Other techniques can change the message conveyed by the visuals, such as including text in a picture to express irony, creation of memes...

Because of this, we should not be surprised by the existance of computing areas that try to exploit this tremendous communication potential. Computer vision is an interdisciplinary field that seeks to automate tasks typical of the human visual system. The creation and growth of this field is closely related to the development of technology: there was no widespread interest in developing algorithms to solve any of the challenges of computer vision without machines powerful enough to implement this algorithms [4]. Computer vision is often considered as a part of artificial intelligence, since it naturally needs its skills when interaction with the environment is mandatory. Other fields commonly associated are: image reconstruction, analysis or transformation in 3D and 2D, pattern recognition, study of biological vision, machine vision to provide robots the ability to automatically inspect their surroundings in real time...

In the last few years, we can appreciate the rise of interest in using feature-based methods (image morphing, panoramic stitching or view interpolation, for example), combined with machine learning techniques and optimization networks, as we will see in the next chapter. In this project we will focus in one of this methods: the neural style transfer. This method uses a convolutional network to extract a certain set of features from two images, and then it combines this features to create an image with a very interesting visual effect. We want to implement the algorithm in order to analyze how it works.

1.2 Objectives and approach

The general problem we are considering is the following: we are given two images I_1 and I_2, and two feature extractors f_1 and f_2 that compute certain aspects of the images (for example, f_1 may extract the color structure, while f_2 may extract shapes). We want to synthesize an image I that "looks like" I_1 according to f_1 and like I_2 according to f_2. Several versions of this problem have been proposed in the literature; the most famous is 'style mapping' in which a painting from a famous artist provides the general 'style' of the synthesized image, and a picture provides the content. We shall put the problem in a general setting and concentrate on texture synthesis, that is, on how to use a texture to create images that can be interpreted as having a well-defined content.

Bearing this in mind, we are setting out to achieve these objectives:

- Give an overview of the state of art in computer vision and specifically in the context of our problem.
- Create a general model that is capable of implementing this 'high level mixing' concept.
- Adapt and specialize this model according to the characteristic to be extracted.
- Experiment with content mixing and advanced style mapping.
- Investigate what happens to the images visually after extracting the features.
- Analyze the behaviour of our method on different cases such as: noise from resizing, image composition, etc
- Study the effects of modifying our control parameters in the results.
- Check our results and compare them to other applications that apply similar transformations.
- Discuss the possible uses of this method in non-artistic approaches.

Our solution to this problem is based on the use of convolutional networks to extract the relevant features of the sample images. Once extracted, we will pose our problem as an optimization one in which, starting from a random image (that is, noise), we will modify its content so that the difference between the sample images and the one we have generated is minimal. In order to do this, we need to define some distances between the content and the features extracted that will be accountable for measuring how close we are to the content or style of the samples.

1.3 Structure of the document

In this very first chapter, we expose the relevance and importance of visual communication and computer vision along with the characteristics of our problem and our goals for the project.

In the second chapter we discuss the techniques used in computer vision (including historical context), recent results and the several approaches to solve the problem. We also detail the technologies used in this project.

In the third chapter we provide a less complex example of feature extraction so we can give the readers a general idea of our solving method. After this example, we explain our proposed solution.

In the fourth chapter we analyze the results obtained after running all the experiments.

Finally, the fifth chapter sums up the results obtained and closes this document giving some thoughts about future work.

2

Style transfer. State of art

The problem of style transfer is one of the newest problems in the field of computer vision. Currently, there is a lot of interest in this area and every year we discover new approaches that improve the current solutions. In this chapter we are going to briefly review the historical development of computer vision from the point of view of our problem, while talking about some of these new approaches. We are also going to describe some details of the techniques comonly used in the style mapping problem and this project.

2.1 History and development of computer vision

In the decades of 1940 and 1950, biologists, cognitive scientists, psychologists and other researchers were studying the field of electrophysiology [5] [6]. Early research took on many of the problems related to brain modeling, including perceptron [7]. This led them to realize that they could describe how the brain parts worked studying the functionallity of the nerves related to them [8]. After several years analyzing samples, they gathered enough information to define the Perceptron in 1958, an algorithm for supervised learning of binary classifiers.

In 1963, Larry Roberts published his Ph.D.thesis at MIT discussing the possibilities of extracting 3D geometrical information from 2D perspective views of blocks (polyhedra) [9]. This publication attracted the interest of the research community, who tried to find a way for the machines to replicate human vision. Researchers quickly realized that it was necessary to work with images from the real world, so they started to develop low-level techniques such as edge detection or segmentation. In 1966, Marvin Minsky assigned to one of his undegraduate students what he thought it would be a simple task: attach a camera to a computer and describe what it was seeing [10]. This turned out not to be an easy task, as they did not have defined any criteria to interpret the data. In 1968, Guzman was able to design a software capable of labelling the lines of complex groups of polyhedral objects [11], while Irwin Sobel and Gary Feldman presented the idea of an "Isotropic 3x3 Image Gradient Operator" at a talk at the Stanford Artificial Intelligence Laboratory [12]. This became known as the “Sobel edge extractor”.

The efforts in the decade of the1970s were focused in developing mathematical tools to improve these techniques. Now the challenge was to explain why vision worked the way it worked, since they already had described the basic computational process in the previous decades. In this decade there was some succcess on interpreting selected images. In 1974, the Special Interest Group on Computer GRAPHics and Interactive Techniques organized its first conference to analyze the improvements in the area. In 1978, David Marr proposed a new paradigm composed by a framework with three levels (computational, algorithmic and implementational) to study, design and implement some possible solutions [4]. This decade formed the foundations of some of the existing algorithms used in computer vision such as: edge extraction from images, labeling of lines, non-polyhedral and polyhedral modeling... We could also see the first commercial applications of computer vision: the Optical Character Recognition (OCR) is released, making typed, handwritten or printed text intelligible for computers.

In the decade of 1980, scientist kept trying to understand human vision and emulate it. The neural networks that were designed by Frank Rosenblatt in the 50s were improved: the processing of an image took place over a number of layers [13]. We can appreciate a shift towards geometry: the mathematical analysis defined the concepts of scale-space [14], a formal theory for handling image structures at different scales, by representing an image as a one-parameter family of smoothed images. This is a fundamental concept in feature detection (edges, blobs, corners...). It was also possible to include texture and shading on surfaces to improve the shape representation, while incorporating powerful tools like Markov random fields to model some of this new structures. Thanks to the development of the techniques, they started to be used for artistic purposes: the films Tron (1982) and The last Starfighter (1984) were the first movies with completely computer-generated shots.

In 1990, a lot of interest was generated by face recognition problems. The first use of statistical methods to recognize faces in images was the eigenface. This concept was developed by Sirovich and Kirby (1987). The eigenvectors that compose the eigenface are derived from the covariance matrix of the probability distribution over the high-dimensional vector space of face images. In 1991, they were used by Matthew Turk and Alex Pentland in face classification tasks by comparing the representation of faces with this method [15].

In this decade we could see an increasing interaction between computer graphics and computer vision, which was accelerated by the Internet. Examples of this interaction include panoramic image stitching, image-based rendering or image morphing, a special effect in motion pictures and animations that changes (or morphs) one image or shape into another through a seamless transition. Researchers also found new applications of previous results, like graph cuts to solve imagen segmentation. In the artistic field, films like Jurassic park (1993) and Terminator 2 (1991) changed the perception about the use of this techniques in the industry. The market increased its interest in incorporating this techniques to increase the appeal of existing products or creating new ones [16] [17].

By the 2000s, the image recognition field has made a great improvement due to the creating of large image datasets available on the Internet. We also saw the first attempts of video processing. New mathematical methods have been incorporated like statistics, optimization or differential equations. The applications are also wider: robotics, security, art... In the following section we will focus on style mapping, one of the most recent applications of the computer vision's techniques in the artistic field.

Style mappping today

In 2014, the Visual Geometry Group at Oxford published a paper that detailed how their neural network family VGGNet worked [18]. A year later, Leon Gatys published the core paper of this project [19]. In this paper, Gatys explained how he used this VGGNets to develop an algorithm capable of extracting high-level features from an image in order to apply it to another image. This method generates an interesting visual output, comonly called 'neural style' and it became popular very quickly.

Since 2015, there have been several papers derived from this, analyzing different configurations or improving the algorithm's performance [20]. Some of them, like Johnson et al. [21] will be studied in the following chapters, but others are very new and their level of complexity leaves them out of the scope of this document: one of them incorporates Markov random fields to smooth the combination of style patches into bigger images [22], while another paper opens the possibility of combining several 'style' features at the same time [23].

The popularity of the technique has propitiated the appearance of web sites and apps that implement it: Prisma, an app that implements the Gatys algorithm using [20] to deliver almost instant results, Ostagram.me, a web site gallery of neural style pictures, or deepart.io, a web site that implements the algorithm and has its own gallery too, are some examples. The topic is still developing: in 2017, a paper was published describing a new way to apply the neural style whose results display such accuracy that it is almost impossible to tell if they are a real picture [24].

It is important to note that the neural approach is not the only technique currently used to change the appearance of a picture: there are some methods that try to modify the histogram of the picture in order to generate a visual output similar to another image [25]. In july of 2015, Google released its DeepDream algorithm using the Inception family of neural networks [26]. The algorithm finds and enhances patterns in images via algorithmic pareidolia, thus creating a dream-like hallucinogenic appearance [27].

2.2 Machine learning

Machine learning is a subset of artificial intelligence that uses statistical techniques to give computers the ability to progressively improve their performance on a specific task with data, without being explicitly programmed. The tasks can be supervised or unsupervised. In supervised learning we have a 'training' set of input data and we know the output that the computer should give to it. The goal then is to make the computer develop some kind of criteria that allows it to replicate the classification and then apply this data to new unclassified inputs [13]. In unsupervised learning we don't know the answer to the inputs, and we want the computer to discover hidden patterns or properties on its own [28].

Machine learning algorithms try to extract patterns from the given data in order to make predictions on it by building models. They are often used when designing and programming explicit algorithms with good performance is difficult or infeasible. It is important to note that when using biased data (such as text, because of context), the machines will incorporate that bias to the model. This factor will be relevant if we try to create a model from those results, because it will introduce errors that will need additional measures to be taken off. Machine learning techniques are used in different approaches such as clustering, bayesian networks, genetic algorithms...

2.2.1 Neural networks

One of the most popular approaches to machine learning is using artificial neural networks. Neural networks are computing systems vaguely inspired by the biological neural networks that constitute animal brains. They are composed by layers of *neurons*. Neurons are functions that take the input, multiply each of them by a number called weight, sum those weighted inputs and adds a numerical term called bias. This number is passed to a nonlinear function called activation and generates an output.

The inputs and outputs of these neurons can be connected within each other. Depending on how the are connected, the neurons will form networks with different properties. If the neurons can be partitioned in groups that can be ordered in such a way that all connections go from a neuron in group n to one in group $n+1$, then the network is called multi-layer, and the groups are called layers. The layer whose inputs are not connected to neurons while the outputs act as inputs for another layer is called the input layer, while the output layer is analogous. All the remaining layers are called 'hidden layers', and their number is arbitrary.

Neural networks receive an input vector $x \in \mathbb{R}^n$. The output of the first layer is passed to the first hidden layer. This layer applies a linear operation with some weight and bias: $y = W * x + b$, where $W \in \mathbb{R}^{m*n}$ is the weight matrix, $b \in \mathbb{R}^m$ is the bias vector and m is the size of the hidden layer. Then a linear fucntion called rectified linear unit (ReLU) is applied to this output (for example, $z = max(0, y)$.

This process is repeated in each hidden layer until we reach the output one. Then, we apply the softmax function, a generalization of the logistic function used in traditional backpropagation. This function takes a K-dimensional vector x of real values into a K-dimensional vector $\sigma(x) \in (0, 1)$ The expression of this function is:

$$\sigma : \mathbb{R}^K \to \left\{ y \in \mathbb{R}^K | y_i > 0, \sum_{i=1}^{K} y_i = 1 \right\}$$

with

$$\sigma(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} \text{ for j} = 1 \dots K$$

This function will be used to interpret the output as categorization: each output neuron represents a category and its value is the probability that the input belongs to that category.

In order to train the networks, that is, adjust the weights and biases to our data, it is very common to use a method called backpropagation. It was originally developed by Paul Werbos in 1974 and called 'ordered derivatives' [29]. It recieved little attention at the time until, in 1987, it was reintroduced by Rumelhart, McClelland and Hinton [30]. The method is used in artificial neural networks to calculate a gradient that is needed in the calculation of the weights to be used in the network.

Backpropagation is a special case of automatic differentiation and is commonly used by the gradient descent optimization algorithm to modify the weights of neurons by calculating the gradient of the loss function. This is also called 'backward propagation of errors', because after calculating the error at the output, it is distributed back through the network layers. In order to use backpropagation, we need to know the derivative of the loss function with respect to the network output, which means that we have a target defined. For this reason, it is considered to be a supervised learning method.

Convolutional networks

A convolutional network is class of deep feed-forward artificial neural networks. They are widely used in recommender systems, natural language processing and image classification tasks due to the fact that the offer similar results to classification algorithms without having to program their behaviour. Since the mathematics of convolutional networks will be covered extensively in the third chapter, we shall limit here to some general architectural considerations.

They have an input layer and an output layer with several hidden layers in between that are used to learn features and do the classification. The final layers are fully connected, that is, every neuron present in a layer is connected to all the neurons from the previous and following layer. This is very costly, as we need to calculate a lot of weights for each neuron even for small sizes. If we tipically want to work with large images, we neet to note that every pixel represents an input variable.

For this reason, most of the hidden layers are convolutional instead. These layers apply a convolution operation to the input, reducing the number of weights required in each layer. Convolutional layers are often paired with pooling layers, which combine the outputs of a set of neurons from the previous layer into a single neuron in the next one.

2.2.2 ImageNet and VGG Networks

ImageNet is an image database organized according to the WordNet hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images. This site organizes a challenge called ImageNet Large Scale Visual Recognition Challenge every year since 2010. The challenge consist in classifiying a large number of pictures from the database into a thousand of categories. The ILSVRC evaluates algorithms for large scale object detection and image classification. This allows researchers to compare progress in object detection across a big variety of objects taking advantage of all the labelling process done in the database's images. The competition also tries to measure the progress of computer vision for large scale image indexing for retrieval and annotation.

In 2012, the results obtained with convolutional networks were so good that drove the attention of the industry. Two years later, the Visual Geometry Group, a research group from the University of Oxford won the first and second places in this challenge with a set of Convolutional Networks designed by them. They achieved 7.5% top-5 error on ILSVRC-2012-val, 7.4% top-5 error on ILSVRC-2012-test [31], and this results results were so impressive that a year later they published a paper explaining the architecture of this networks [18]. Since then, they have been improved and have set the standard in visual recognition.

As we can see in the paper and in the figures below, the VGG neural network is an image classification convolutional neural network. Given an image, the VGG network will output probabilities of the different classes that an image could potentially belong to. For example, the VGG network might output values of 0.93 for cat and 0.02 for flower, indicating that there is a 93% chance (i.e it is 93% confident) that the image contains a cat and that there is a 2% chance that the image contains a flower. In Figure 2.1, we can see the architecture of the VGGNets.

In Figure 2.2 it is possible to observe a comparison between the sizes of the VGG16 layers and the original image. All the layers that have the same size, along with the inmediate following pooling layer constitute what we will call a block.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2.1: VGGNet Architectures. The 16-layer version is highlighted

The reason why this is relevant for our project is because this networks are already pre-trained and publicly available in many frameworks to do classification tasks. However, as shown in Gatys paper, they can be used as a feature extractor only if we drop the last layers that are in charge of classifying. In this project we will use the VGG16 version instead of the VGG19 one that is used in Gatys because they offer very simmilar accuracy but VGG16 is a little bit smaller, speeding up our solution.

Machine learning frameworks

Regarding to the use of neural networks and all its variants, there is a huge variety of tools to implement and manage them: frameworks, libraries, toolkits... One of the most popular ones is Torch, a framework written in the scripting language Lua. It is very easy to install and use and has helpful error messages. Another popular example is Theano, developed at the University of Montreal. Theano works at a very low-level and it is a little bit difficult to install and configure. However, it displays a very good performance. Both frameworks are widely used in researching.

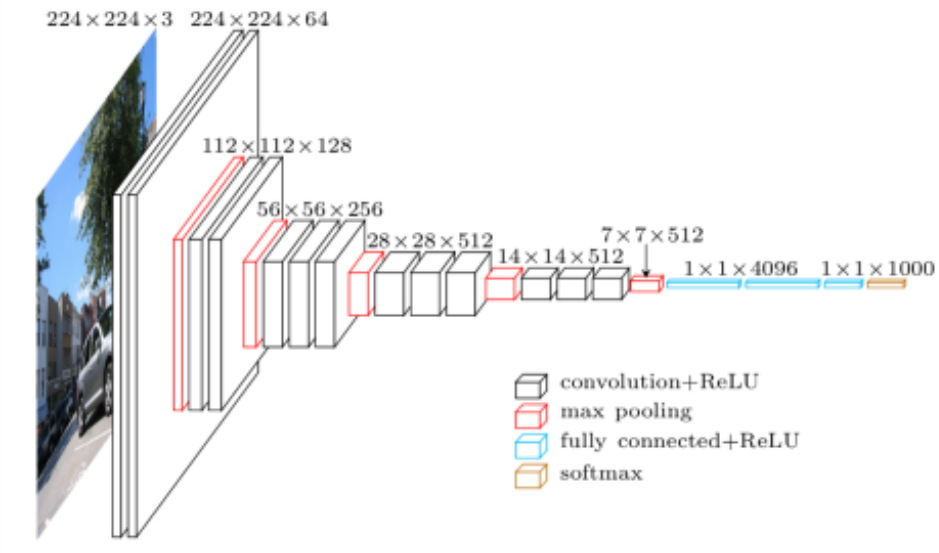


Figure 2.2: Visual representation of the VGG16 Pre-Trained Model

We also have TensorFlow, a framework developed by Google a bit slower than Theano a Torch. TensorFlow is often described as a modern version of this two that incorporates its experience over the years. It has a lot of tutorials to quickly learn how to use it and runs in several platforms.

On the one hand, there are frameworks developed specifically for image classification/machine-vision leveraging convolutional neural networks like Caffe, which has a set of pre-trained models called Model Zoo. On the other hand, we have examples like CNTK (Microsoft Cognitive Toolkit), that is best suited to work with sound or texts instead of images. In conclusion, there is a lot of variety in this area and it is constantly improving, which is a good sign of interest in this problems and eases finding new and better soutions.

2.3 Optimization

In mathematics, computer science and operations research, mathematical optimization is the selection of a best element from a set of elements. Formally, given a function $f : A \to \mathbb{R}$ from some set A to the real numbers, we look for an element x_0 in A such that $f(x_0) \leq f(x)$ for all $x \in A$ (minimization) or $f(x_0) \geq f(x)$ for all $x \in A$ (maximization). Tipically, the A set is defined by some constraints and its elements are the candidate solutions, while the function f is called cost function, loss function or other names depending on the problem.

In this field we may just try to know if there exists any solution under our constraints, but most of the time we have defined some kind of criteria to find the best one according to them. There are several methods to find this solutions, and most of them rely on iterative process to refine a solution until we reach an optima. This algorithms are used across many fields such as economics and finance, engineering, mechanics...

There is a wide variety of methods to try to find solutions to this problems. Most of them study the gradient or Hessian matrix to find minimum and maximum points, while using techniques such as Lagrange multipliers or Kuhn-Tucker conditions [32] to turn constrained problems into unconstrained ones. However, while this improves the convergence of the methods, this objects can be very costly because of computational complexity. Some examples of this kind of methods are the 'Simplex Algorithm' or the combinatorial ones.

In order to avoid calculating the Hessian matrix and other objects, some methods try to approximate them by other means. For example, Newton's method or interior point methods can approximate Hesians using finite differences, while pattern search and interpolation methods evaluate only funciton values and can use finite differences when the funcitons are continuously differentiable.

Other methods try to approximate gradients, like the ellipsoid method or the Simultaneous perturbation stochastic method (SPSA). In this project we will focus in *gradient descent*, a first-order iterative optimization algorithm for finding the minimum of a function and *L-BFGS*, the limited-memory version of the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [33]. This is a quasi-Newton method that uses an estimation of the inverse Hessian Matrix. Quasi-Newton methods are generalizations of the secant method to find the root of the first derivative for multidimensional problems and they come in handy when we are dealing with medium-large problems ($N < 1000$) and the Jacobian or Hessian are unavailable or too expensive to compute.

2.4 Libraries and other tools used in this project

In order to implement the proposed solution and do our experiments, we have written a program in *Python*. Python is an interpreted high-level programming language for general-purpose programming. It supports multiple programming paradigms and is available for many operating systems. It also has a lot of libraries implemented by the comunity, which makes Python a very interesting choice because of its flexibility and abundance of tools. We have used the module *numpy* to perform all the operations with arrays and a module to manage images (*Image* in our case). We have also used the module *scipy* because it implements the L-BFGS algorithm. To do all the tasks related to the control and management of the convolutional network, we need a machine learning framework such us TensorFlow, Theano, Caffe... We have chosen *TensorFlow* to do all the tasks related to the control and management of the convolutional network because it is very easy to install and does not require additional configuration to work.

Each of this frameworks has its own implementation of the VGGNets, because the internal details of the architecture change from one to another, so the frameworks have their own APIs to work with the networks. For this reason, we have used a library from Python called *Keras*. Keras is a high-level neural networks API, capable of running on top of this frameworks. It provides the necessary level of abstraction from the convnet implementation and a common API for all the frameworks that are compatible with it (currently TensorFlow, Theano and CNTK). We need the following modules from Keras: 'Backend' to comunicate Keras with the tensor manipulator library (TensorFlow for us), 'Model' to do operations with the VGG layers, and 'VGG', the one that contains the network implementation, to use it as our convnet.

We have also written a little example with some constraints to ease the understanding of our project. The language has been written in MATLAB, an IDE with its own language called M. This language is optimized to do numerical computing, performing matrix manipulations, plotting of functions and data and other features in a very efficient way. Regarding to the supporting role, we have used *Anaconda* to install Python and its modules in Windows. We have also used the Python's notebook system, *JuPyter*, as our testing interface.

3

Feature extraction

3.1 General method explanation

In this chapter we introduce the main concepts behind the implementation of our algorithm. After that, we provide a simplified example that illustrates how the process works: we extract the edges with the Sobel technique and use simple gradient descent to get the results. Finally, we show how to perform all the tasks in a general case.

3.1.1 General ideas

Let us assume that we have two images I_1, I_2, and that we want and image with the same contents as I_1 (that is, the same composition, spatial distribution, objects, shapes...) and the same style as I_2 (same colors, same textures, technical details like brush strokes...). We have two feature extractors: the content extractor ϕ and the color structure and texture extractor ψ.

We look for an unknown image I such that $\phi(I) \sim \phi(I_1)$ (this means that I has, more or less, the same content as I_1) and $\psi(I) \sim \psi(I_2)$ (I has, more or less, the same color structure as I_2). In general, the more $\phi(I)$ will approach $\phi(I_1)$, the more $\psi(I)$ will be different from $\psi(I_2)$, especially if ϕ and ψ have some sort of interaction and we can't change one without modifiying the other.

The standard way to solve this kind of cunundrum is by a weighted combination of errors. Let us assume that ϕ and ψ are vector-valued functions: $\left(\phi : \mathbb{R}^{N*M*C} \to \mathbb{R}^P, \psi : \mathbb{R}^{N*M*C} \to \mathbb{R}^Q\right)$. We define the weighted error for an image I as:

$$E(I) = \frac{(1-\alpha)}{2}\|\phi(I) - \phi(I_1)\|^2 + \frac{\alpha}{2}\|\psi(I) - \psi(I_2)\|^2,$$

where $\|\cdots\|^2$ is the Euclidean distance in the suitable space ($\mathbb{R}^P$ or $\mathbb{R}^Q$) and $0 < \alpha < 1$ is a weight parameter that determines the relative importance of the error on ϕ versus the error on ψ. Our problem is how to find an image I that, given I_1 and I_2 minimizes $E(I)$.

If ϕ and ψ are continuous and continuously derivable (C^1), then E is also (C^1), so we can use standard optimisation methods such as gradient descent. We start with a random image I_0 (white noise image), compute the gradient $\nabla E(I_0)$ and move I_0 to I_1 with a small step in the direction in which the gradient 'descends', that is, towards smaller values of E.

This will be the general iteration:

$$I_t - \gamma \nabla E(I_t) \to I_{t+1},$$

where γ is a (smallish, for stability) step parameter. The iteration reaches a stable point when $\nabla E = 0$, corresponding to a minimum of E (in theory, $\nabla E = 0$ could be a maximum, but the iteration is unstable around maxima and never reaches an equilibrium then).

Discrete gradient computation can be noisy, sometimes causing the iteration to wander around the same area aimlessly. To avoid this, one often introduces a momentum term, which keeps the iteration from changing direction too quickly, acting effectively as a low-pass filter on the gradient:

$$\mu_{t+1} = (1-\nu)\mu_t + \nu \nabla E(I_t),\ 0 < \mathrm{v} < 1,\ \mu_0 = 0$$
$$I_{t+1} = I_t - \gamma \mu_t$$

For $\nu = 1$ we obtain the original undamped iteration. For $\nu = 0$, the method doesn't work, as the image never moves from the original point I_0. The gradient is composed of the derivatives of E with respect to the elements I^c_{ij} of the image, where i, j are the coordinates of a pixel and c is the color channel. This is given by:

$$\frac{\partial E}{\partial I^c_{ij}} = (1-\alpha)\|\phi(I) - \phi(I_1)\| \frac{\partial \phi}{\partial I^c_{ij}} + \alpha \|\psi(I) - \psi(I_2)\| \frac{\partial \psi}{\partial I^c_{ij}}$$

where

$$\frac{\partial \phi}{\partial I^c_{ij}} = \left[\frac{\partial \phi^1}{\partial I^c_{ij}} \cdots \frac{\partial \phi^d}{\partial I^c_{ij}} \right]'$$
$$\frac{\partial \psi}{\partial I^c_{ij}} = \left[\frac{\partial \psi^1}{\partial I^c_{ij}} \cdots \frac{\partial \psi^q}{\partial I^c_{ij}} \right]'$$

The derivatives $\frac{\partial \phi^k}{\partial I^c_{ij}}$ and $\frac{\partial \psi^k}{\partial I^c_{ij}}$ depend on the specific form of the feature extractors. One important case for application due to its widespread acceptance these days is that in which our iteration is the output of one of the layers of a convolutional network. This case we shall now analyze.

3.1.2 Relevance regarding our experiments

We can use this schema to analyze one feature at a time. If we want to study $\phi(I)$, for example, we only use the terms with ϕ in the previous equations and we build an image I' such that $\|\phi(I') - \phi(I)\|^2$ is minimal, which means that I' has only the essential. After that, we define $E(\phi) = \|\phi(I') - \phi(I)\|^2$ and do the iteration $I_t - \gamma \nabla E(I_t) \to I_{t+1}$

3.1.3 Deep Neural Networks

A feed-forward neural network is a cascade of layers, each one taking as an input the output of the previous layer and applying a function to it. The results of this funciton will shape the input of the next layer and so on until the last layer. This is represented in the following schema:

$$I \xrightarrow{z^0} f_0 \xrightarrow{z^1} f_1 \cdots \xrightarrow{z^{n-1}} f_{n-1} \xrightarrow{z^n} \phi(I)$$

We have defined an error function $E(z^n)$ and we want to determine the derivative of E with respect to the components z^0 , $\nabla_0 E$. We assume $z^k \in \mathbb{R}^{u_k}$ and $f_k : \mathbb{R}^{u_k} \to \mathbb{R}^{u_{k+1}}$. We can write

$$\frac{\partial E}{\partial z_i^0} = \sum_{j=1}^{u_n} \frac{\partial E}{\partial z_j^n} \frac{\partial z_j^n}{\partial z_i^0}$$

The derivatives $\frac{\partial E}{\partial z_k^n}$ are usually easy to compute given the function E, so the problem is reduced to computing the derivatives of the network's outputs with respect to the inputs: $\frac{\partial z_k^n}{\partial z_i^0}$.

We do this by propagating the derivatives though the layers from output to input with backpropagation. The idea is to compute $\frac{\partial z_k^n}{\partial z_h^\mu}$ as a function of $\frac{\partial z_k^n}{\partial z_h^{\mu+1}}$, where the Greek indices span the layer sequence. We can do the backpropagation with the following equations:

$$\frac{\partial z_k^n}{\partial z_h^\mu} = \sum_{i=1}^{u_{\mu+1}} \frac{\partial z_k^n}{\partial z_i^{\mu+1}} \frac{\partial z_i^{\mu+1}}{\partial z_h^\mu} = \sum_{i=1}^{u_{\mu+1}} \frac{\partial z_k^n}{\partial z_i^{\mu+1}} \frac{\partial f_i^{\mu+1}}{\partial z_h^\mu}$$

The last part of the formula is very useful in the case of neural networks in which the input is first transformed linearly and then the same function f is applied to all the resulting components.

In the case of modern neural networks applied to images, we have three types of layers: convolution layers, sub-sampling layers and fully connected layers. Fully connected layers take as input a vector in $\mathbb{R}^n$, multiply it by a $\mathbb{R}^{m*n}$ weight matrix w and apply a non-linear function f to each component of the result, that is, if the vector x is the input and z is the output: $z_i = f\left(\sum_{k=1}^n w_{ik} x_k\right) = f(c_i), i = 1, ..., m$. In this project, however, we will not use the fully connected layers of the final stages of our convolutional network.

Convolutional layers operate on rectangular arrays of vectors. The input $x \in \mathbb{R}^{N*M*D}$, that is, each 'pixel' x_{ij} is an element of $\mathbb{R}^D$. Similary, we have $y \in \mathbb{R}^{N*M*Q}$. These layers do not change the dimmension of the array (both input and output are $M * N$ arrays), but they may change the dimensionality of each 'pixel' from D to Q. The convolution is performed by masks of size $(2m + 1) * (2m + 1)$ (in many networks $m = 1$). Each activation c_{ij}^d depends on all the components of the input vectors in a $(2m + 1) * (2m + 1)$ neighbour around it. That is, given $\xi, \zeta \in \{-m...m\}$, we have

$$c_{ij}^d = \sum_q \sum_{\xi,\zeta} w_{\xi\zeta}^{qd} x_{i+\xi,j+\zeta}^q, i = 1, ..., N; j = 1, ..., M; d = 1, ..., D$$

The input can be padded with zeros or other techniques in order to control the spatial dimensions in the convolution. It is important to note that because this filter doesn't change as we apply it across the image, if a feature generates an activation, it will do it in any spatial position.

The output is obtained by applying a non-linear function to the activators:

$$y_{ij}^d = f\left(c_{ij}^d\right) = f\left(\sum_q \sum_{\xi,\zeta} w_{\xi\zeta}^{qd} x_{i+\xi,j+\zeta}^q\right) i = 1, ..., N; j = 1, ..., M; d = 1, ..., D$$

The derivatives that we need to calculate are:

$$\frac{\partial}{\partial x_{ij}^q} y_{i-\xi,j-\zeta}^d = f'\left(c_{i-\xi,j-\zeta}^d\right) w_{\xi,\zeta}^{qd} = f'\left(\sum_q \sum_{\alpha,\beta} x_{i-\xi+\alpha,j-\zeta+\beta}^q w_{\alpha,\beta}^{qd}\right) w_{\xi,\zeta}^{qd}$$

This function f was traditionally the sigmoid function $\sigma(x) = (1 + e^{-x})^{-1}$, but this is not a good choice for deep networks with many layers: each layer, the derivatives are multiplied by $\sigma\prime(x) = \sigma(x)(1 - \sigma(x))$, which is very small for most values of x. As a consecuence, repeated multiplications result in very small values of the derivatives. Nowadays the following function is preferred to be used with deep networks:

$$u(x) = \begin{cases} x, \text{if x} > 0 \\ 0 \text{ otherwise} \end{cases}$$

The propagation of ∂E through the layer can be calculated as:

$$\frac{\partial E}{\partial x_{ij}^q} = \sum_d \sum_{\xi,\zeta} \frac{\partial E}{\partial y_{i-\xi,j-\zeta}^d} \frac{\partial y_{i-\xi,j-\zeta}^d}{\partial x_{ij}^q} = \sum_d \sum_{\xi,\zeta} \frac{\partial E}{\partial y_{i-\xi,j-\zeta}^d} u\left[c_{i-\xi,j-\zeta}^d\right] w_{\xi,\zeta}^{qd}$$

Sub-sampling layers reduce the dimensionality of the input matrix by max-pooling. This process creates groups of four pixels from the input that are transformed into a pixel in the output by taking the componentwise maximum of the four vectors: $c_{ij}^q = max\{x_{2i,2j}^q, x_{2i,2j+1}^q, x_{2i+1,2j}^q, x_{2i+1,2j+1}^q\}$. Max-pooling is the way sub-sampling layers actually work in deep networks. However, for reconstruction it is often easier to pretend they work by averaging the input values: $c_{ij}^q = \frac{1}{4}\{x_{2i,2j}^q + x_{2i,2j+1}^q + x_{2i+1,2j}^q + x_{2i+1,2j+1}^q\}$. This hypothesis 'spreads' better the derivative on the whole input, leading to better backpropagation.

Regardless of the model used, this layer implements a function $f : \mathbb{R}^{N*M*Q} \to \mathbb{R}^{\frac{N}{2}*\frac{M}{2}*Q}$. In this case, the size of the matrix is changed while the dimensionality of the 'pixels' is unchanged.

To compute the derivative in the first case, set

$$l(x) = \begin{cases} \{i, i+1\}, \text{if i is even} \\ \{i-1, i\}, \text{if i is odd} \end{cases}$$

and $Q(i,j) = l(i) * l(j)$. Given an element x_{ij}^q , the set $Q(i,j)$ is the set of indices of elements with which x_{ij}^q is compared to determine the maximum. Therefore

$$\frac{\partial y_{\lfloor \frac{i}{2} \rfloor, \lfloor \frac{j}{2} \rfloor}^q}{\partial x_{ij}^q} = \begin{cases} 1, \text{ if } x_{ij}^q = \max_{k,h \in Q(i,j)} x_{kh}^q \\ 0 \text{ otherwise} \end{cases}$$

In the second model,

$$\frac{\partial y_{\lfloor \frac{i}{2} \rfloor, \lfloor \frac{j}{2} \rfloor}^q}{\partial x_{ij}^q} = \frac{1}{4}$$

In this case:

$$\frac{\partial E}{\partial x_{ij}^q} = \frac{1}{4} \frac{\partial E}{\partial y_{\lfloor \frac{i}{2} \rfloor, \lfloor \frac{j}{2} \rfloor}}$$

3.2 Basic model example: Edge extractor

In this section, we only work with a sample image, the famous picture known as 'Lena' (figure 3.1a). On the one hand, we used a black and white version of the picture (fig. 3.1b) to reduce the dimensional space that we were working on. Then we extracted the edges of this picture using a Sobel operator. After that, we calculated the gradient approximation of the image and its direction. On the other hand, we applied the same process to a random image, and we calculated the error as the difference of gradients. This way, we modified the random image according to this error to generate a new image and applied the same process to it iteratively.

(a) Lena (RGB) (b) Black and white version

Figure 3.1: Both versions of Lena

On the one hand, we have the picture of Lena, called $\overline{I}$ for the purposes of this example. This image has been transformed into a black and white version to avoid working with the colour layers. We have applied a gaussian filter to reduce the impact of noise and to ease the location of features.

The Sobel operator uses two kernels:

$$A = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix},$$

$$B = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

When these kernels are convolved with the image $\overline{I}$, the results are two images U, V that contain the vertical and horizontal edges of the image respectively. The mathematical expressions for these operations are:

$$\overline{u}_{ij} = \sum_{\alpha,\beta} \overline{I}\left[i+\alpha, j+\beta\right] * A\left[\alpha,\beta\right]$$

$$\overline{v}_{ij} = \sum_{\alpha,\beta} \overline{I}\left[i+\alpha, j+\beta\right] * B\left[\alpha,\beta\right]$$

where $\alpha, \beta \in \{-1, 0, 1\}$. This way, we can calculate the gradient of the image as: $\overline{m}_{ij} = \sqrt{\overline{u}_{ij}^2 + \overline{v}_{ij}^2}$. We can also calculate the gradient's direction as: $\overline{\theta}_{ij} = \tan^{-1} \frac{\overline{v}_{ij}}{\overline{u}_{ij}}$. At this point, we need to apply padding (we chose zero padding) to the image to make up for the pixels lost after convolving.

On the other hand, we would like to find an image that has the same gradient and gradient's direction as $\overline{I}$. However, it is not easy to generate an image like that from scratch. For this reason, we are going to generate a random image I and modify it following this steps: We apply the previous method to I to calculate its gradient and the gradient's direction, that is, M and θ. We define the error function as $E = \frac{(1-\nu)}{2}\|\overline{M} - M\|^2 + \frac{(\nu)}{2}\|\overline{\theta} - \theta\|^2 = (1-\nu)\,P + (\nu)\,Q$

We define the iteration $I_{ij} - \gamma\frac{\partial E}{\partial I_{ij}} \xrightarrow{\text{t to (t+1)}} I_{ij}$. In order to calculate $\frac{\partial E}{\partial I_{ij}} = (1-\nu)\,\frac{\partial P}{\partial I_{ij}} + (\nu)\,\frac{\partial Q}{\partial I_{ij}}$, we need to calculate:

$$\frac{\partial P}{\partial I_{ij}} = \sum_{\alpha,\beta}\left(\overline{m}_{i-\alpha,j-\beta} - m_{i-\alpha,j-\beta}\right)\frac{\partial m_{i-\alpha,j-\beta}}{\partial I_{ij}}$$

$$= -\sum_{\alpha,\beta}\frac{\overline{m}_{i-\alpha,j-\beta} - m_{i-\alpha,j-\beta}}{\sqrt{u^2_{i-\alpha,j-\beta} + v^2_{i-\alpha,j-\beta}}}\left(u_{i-\alpha,j-\beta}\frac{\partial u_{i-\alpha,j-\beta}}{\partial I_{ij}} + v_{i-\alpha,j-\beta}\frac{\partial v_{i-\alpha,j-\beta}}{\partial I_{ij}}\right)$$

$$\frac{\partial Q}{\partial I_{ij}} = -\sum_{\alpha,\beta}\frac{\left(\overline{\theta}_{i-\alpha,j-\beta} - \theta_{i-\alpha,j-\beta}\right)}{\sqrt{1 + \left(\frac{v_{i-\alpha,j-\beta}}{u_{i-\alpha,j-\beta}}\right)^2}}\left[\frac{1}{u_{i-\alpha,j-\beta}}\frac{\partial v_{i-\alpha,j-\beta}}{\partial I_{ij}} + \frac{v_{i-\alpha,j-\beta}}{u^2_{i-\alpha,j-\beta}}\frac{\partial u_{i-\alpha,j-\beta}}{\partial I_{ij}}\right]$$

where $\frac{\partial u_{i-\alpha,j-\beta}}{\partial I_{ij}} = A(\alpha,\beta)$, $\frac{\partial v_{i-\alpha,j-\beta}}{\partial I_{ij}} = B(\alpha,\beta), \alpha,\beta \in \{-1,0,1\}$.

The stop condition of the iteration will be $\|I_{ij}(t+1) - I_{ij}(t)\|^2 < \epsilon \equiv \|\gamma\frac{\partial E}{\partial I_{ij}(t)}\|_\infty{}^2 < \epsilon$ or after a certain number of iterations.

Figure 3.2 shows the results of this approach. As we can see, the borders in the obtained figure are pretty close to the original, as we can clearly recognize the hat, hair, right shoulder, eyes, noise and mouth of Lena. We can not see many details because of the limitations of the technique: we can not clearly distinguish the feathers of the hat or the reflection in the mirror.

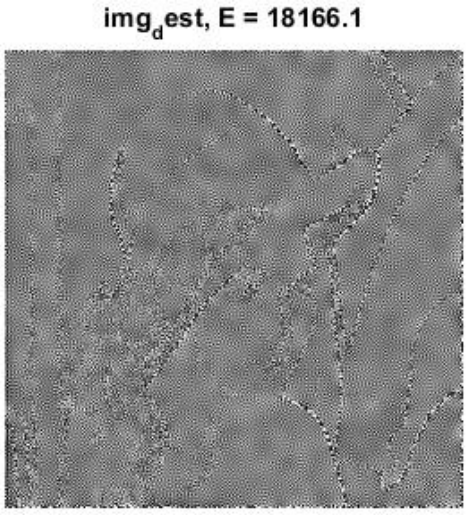


Figure 3.2: Gradient descent results: Lena's approximation

3.3 Style transfer problem

In the previous section, we have been able to write an explicit method to extract the edges of an image and implement it. This becomes an impossible task if we want to extract a feature and we do not know the associated transformation. It also makes very difficult to extract several features at the same time. This is the reason why we need machine learning: we do not need to make the extraction of the features by hand.

Our situation has changed, because now we work with two images: the image called C is the one that possesses the structural and spatial information, while the image called S has the colour and textures that we want to apply to the content from C. The result will be the style-transferred image X. This can be posed as an optimisation problem:

$$x^* = argmin_x \left(\alpha L_{content}(C, X) + \beta L_{style}(S, X)\right)$$

Here we are using Gatys [19] method's nomenclature for $L_{content}(C, X)$ and $L_{style}(S, X)$ where C is the photograph p and S is the artwork a.

Note that we need some kind of artifact to encode the semantic information related to content and style. In order to do that, we are going to use part of a classifier based on deep convolutional neural networks: VGGNet. As we previously mentioned in chapter two, this network is already pre-trained with millions of images from the ImageNet project. Instead of writing a new convnet from scratch, we are going to take advantage of this fact and use the model that won the ILSVRC in 2014. The main highlight is that this convnet is so well trained to recognize and classify images that the features it extracts are invariant to all image variation that is not part of the object's identity.

Any further details can be found in the following section, which describes the main highlights of our proposed solution.

3.3.1 Explanation of the process

The first step is to load C and S. If they do not match our preferred size (512 x 512 in our example), we will need to resize them. This simplifies the process, because we would need to split the picture into smaller ones, apply the process to each patch and then put back together all the pieces. Now we reshape the images to add an additional dimension. This will be useful later when we want to concatenate the representation of the images to other structures.

We need to do some additional transformations [18]: we need to substract both images the mean RGB value from each pixel (R = 103.939, G = 116.779, B = 123.68) and change the order of the layers from RGB to BGR. With this steps, we can define three Tensorflow graphs: one for each C, S and a placeholder for X. These graphs can be thought as special variables of TensorFlow that will contain the data of the pictures and some additional information to perform the machine learning tasks.

Then we concatenate this variables into a tensor, which is a generalization of vectors and matrices to potentially higher dimensions. Internally, TensorFlow represents tensors as n-dimensional arrays of base datatypes. With this steps, we have created the structure that will be used as the input of the convolutional network model.

As a feature extractor, we use the VGG16 version [21] instead of VGG19 [18] because both give simmilar results but VGG16 is faster (it has fewer layers). We do not need to use the layers called 'FC-4096', 'FC-1000' and 'soft-max', because they are the ones in charge of the clasification problem. We only use the concolutional layers, as we mentioned in chapter two.

We need to find a way to measure the semantic difference between objects. In order to extract the style, we calculate the Gram Matrix of a certain set of layers [34]. In linear algebra, this is the Hermitian matrix of inner products of a set of vectors $v_1, \ldots, v_n$ in an inner product space given by $G_{ij} = \langle v_i, v_j \rangle$. It allows us to measure the linear independence between features, since the terms of this matrix are proportional to the covariances of the set, telling us which features tend to activate together. The process is calculated between the layers 'block1_conv2', 'block2_conv2', 'block3_conv3', 'block4_conv3', 'block5_conv3' as defined in Johnson et al. The content of the layers is obtained from the model, which was loaded with the tensor structure previously mentioned. With the gramian matrix we can calculate the style loss between two layers as: $\frac{1}{4*N*M} \sum \left(S_{ij} - {C_{ij}}^2\right)$, where $N = 3$ is the number of channels and $M = height(S) * width(S)$.

We need to introduce another parameter called total variation. This is a normalisation term that reduces the noise in X by encouraging spatial smoothness [35]. It is a common practice to include this anisotropic diffusion technique, also called Perona-Malik diffusion, as it reduces the image noise without removing significant parts of the image content. It can be calculated with an anisotropic diffusion; since images are discrete ($\mathbb{R}^{HxW}$), we can use a finite-difference approximation: $TV(X) = \sum_{i,j} \left((x_{i,j+1} - x_{ij})^2 + (x_{i+1,j} - x_{ij})^2\right)^{\frac{\beta}{2}}$. We have chosen $1 < \beta = \frac{5}{2} < 2$, penalising large gradients and distributing changes across regions rather than concentrating them at a point or curve. This will not completely remove the spikes caused by max pooling.

The content loss is the scaled squared Euclidean distance between C and X: $\sum \left(X_{ij} - {C_{ij}}^2\right)$. The data used to calculate this difference is extracted from a single layer of VGG16. Gatys et al. uses 'block4_conv2', but is not the only choice; we see 'block2_conv2' being used by Johnson et al. All three content, style and total variation losses are multiplied by a control parameter that allow us to modify the impact of each component in the resulting image.

Finally, we need to define the gradients of the total loss relative to X and use the gradients to modify it and minimize the loss. The image X begins as a random set of pixels and is improved after every iteration of the gradient descent algorithm. We are not using the standard version but the L-BFGS one. This is because the standard can be unstable [36] and needs a huge number of iterations to converge. With L-BFGS, most of the time we do not even need more than ten iterations to see that the loss stops reducing significantly while giving us a good looking output (although the time for each interation is bigger than the time for a standard iteration, it is compensated). The last step is to reshape the array into an image again and reverse the RGB transformation that we did previously.

4

Experiments, tests and results

4.1 Introduction

In this chapter we discuss the results of our research and the experiments performed. Firstly, we do an analysis on how the features are extracted from an image and the changes that happen during the process. Secondly, we show our results on content mixing from two images. Finally, we talk about style and content mixing and show the executed tests.

4.2 How does the convolutional network work

As we will see in the following analysis, VGG16 has five convolutional + ReLU blocks, each followed by a max-pooling layer. We have dropped the fully connected layers and the softmax one. The convolutional blocks are divided into several layers. The first ones contain a lot more information than the last ones: there is a factor two of difference between the sizes of the first four blocks respectively and a factor four between the fourth and the fifth.

In the first block, the convnet keeps almost all the information from the image. The second block separates the background from the main focus of the picture and starts to loose information about the color. The third block identifies the main object of the image, the one that we would get as the result from the softmax layer if we wanted the classification. In the fourth block the colour is definitely lost, and we can appreciate some heavy losses of accuracy regarding to the shapes of the objects. Finally, the fifth block retains only the most significative edges of the objects, but at this point we barely see some of the original features of the image.

4.2.1 Analysis of a simple image

Block 1 and block 2

As we can see, the first block of VGG does not produce any visual changes in the image, but its work is very important since it acts as the pre-processor for the next blocks (Figures A.2, A.3)

In the second block we have two convolutional levels like in the first block. In the first one (figure A.5), we can appreciate that flat zones of the image have some noise in them. Notice the change in the background color as it presents a reddish zone next to the purple triangle. If we take a closer look, we can see that the background is composed by light blue, yellow and light pink/purple pixels.

If we look at the inside part of the figures, we can see that is affected too. A pattern simmilar to scales or a mosaic can be observed, caused by the appearance of multiple pixels of different colours: yellow with orange and green ones, orange with red and green ones, green and red with blue... The purple and blue triangles are striking because the pattern is caused by different shades of each colour instead of a different one.

Lastly, we can see that the black borders are losing the typical shadow caused by multiple pixels of gray. In the previous block, and probably caused by the errors, we could see that some of this gray pixels went missing. In this block we can notice the appearance of small gaps in the edges of the figure (left border in the green rectangle) close to the places where this pixels dissapeared. The external zones of the black areas also start to lose definition as they start to mix with the colours of the external areas.

The second convolutional level displays a more green appearance (figure A.6). The pixel noise described in the previous level is increased here and it provokes the visual effect of flatten the image. This is because the colour zones can not be percieved as completely separated from another. It becomes more clear now that the figures in the image retain a kind of white area surrounding them while the background is specially affected by the green color.

Conclusion

In conclusion, in this block we can see that the network begins to extract the colour from the image. The process is very quick in the background and it appears to encircle the shapes in the image. This behaviour will help to locate the shapes in the image, as they mostly retain its borders and colour. We can also tell that the red color and the ones close to it are the first to be affected, followed by green.

Block 3

This block affects the color in the internal part of the shapes. We can see (figures A.8, A.9, A.10) how the sort of white zone around the polygons is lost progressively in every level. Also, the color in the background starts to homogenize into a grey color. However, if we take a closer look, the dominant colors in the background are blue, green and purple in different tones.

The color in the borders also dissapears in some parts of the inside. This causes loss of accuracy in the contours and holes in them. The figures start to show big color holes too: this process starts with the red color first and is followed by mostly green and yellow. It can be observed how the red figures gain a hole in its center in level two while the yellow and green ones get the holes in level three.

Conclusion

The color loss intensifies, and starts to affect all the objects in the image. The edges of the objects lose accuracy.

Block 4

In the first level of this block (figure A.12) the contours of the figures begin to look as if the were handrown due to the holes and lack of precision from the previous level. The 'noise pixels' from the previous blocks in the background soften progressively creating little areas of purple and green mixed with blue all over the picture, including the background and the polygons.

From level one to three the color areas in the inside of the figures almost disappear completely (fig. A.13 A.14). We can only see some reminiscent zones of colour in the closest areas to the black borders. The thick border lines are also completely missing and now we can only see very thin residual lines from the edges of the borders.

Conclusion

To sum up, levels two and three seem to perform a simple gaussian blur that enlarges the 'noisy' zones of pixels. The effect creates an amalgam of the colors of purple, green and blue, probably a result of the red colour mixing with little samples of blue that are later combined with the green ones. This mixing increases the number of blue pixels, making them the most common along with purple and green. This phenomenom causes the progressive dissapearance of the borders of the shapes too, making them blend with the background. For this reason, we can assume that contours look at this level less well-defined. Objects and textures of little sizes will probably dissapear or mingle between them, leaving only the biggest shapes.

Block 5

In the first level (figure A.16), the original color is almost completely lost and we can only see the expansion and homogenization of purple and green expand. Additionally, we can also observe a funny effect in the borders: the external edges of the figure and the internal ones are no longer recognized as part of the same border because of the lack of color. In the second level (figure A.17), this pattern generates two different polygons, one inside the other and both presenting a very vague while familiar silhouette. In the last level its possible to see some of the remaining edges, specially the triangled ones, but pretty much all the content is mingled and is not possible to see any futher details.

We can infer that this level extracts the remaining contours of the objects. From level two to level three (figure A.18), we can appreciate again another heavy gaussian-like effect that leaves us with the very edges of the objects. Since the shapes in this picture are so simple, the only edges remaining are the vertex and the straight ones from the polygons while the circle's one is practically missing. This make us assume that building shapes or similar objects would still be distinguishable, while things like the horizon line or fur patterns would be missing unless they are the focus of the picture, so they would be big enough to resist through all the process.

Conclusion

As we can see in the pictures, the layers of this block keep very little information of the original picture. We can see that only very clear edges are preserved, and those are not very accurate compared with the originals.

4.2.2 Analysis of a complex image

Block 1 and block 2

Like in the previous example, we can not appreciate any relevant changes in the image after being processed in the first block aside from some little noise, almost imperceptible unless we zoom it, in the background (figures A.21, A.22). This behaviour was present in the previous example, and we can attribute it to the algorithm or as an effect derived from the convnet. It is important to notice that there are two flowers in the image, but we will be refering to them as a single one unless stated otherwise. This is because the flowers are too close and it is difficult to draw a line to separate which part belongs to each one.

In the second block we see again the same pattern: some red artifacts appear in the plain background areas without any object. In the second layer (figure A.25), the background changes very aggressively to green. We can also observe the remaining original black background surrounding the flower. Interestingly, this detail can not be seen in the green stem part in the lower right section of the image. This indicates that the network is able to identify the main shapes in the image with this process, and the stem is not in contact with the flower like the part in the middle left area. We can theorize that the flattening effect begins at the edges of the picture and then expands though the image until a big border appears. That is identified as object, which leaves the characteristic white area around it.

Regarding to color, we can see again the same color contrast from level one to level two: first the empty background area gets some red spots and then the mossaic effect intensifies and turns everything green. Comparing this results to the previous example, we can now tell that all the figures were interpreted as a big single object, that is why we only saw the background behaviour in the big right empty area and not between the shapes. The convnet identifies the big shapes at this level but not the objects individually. This probably helps it to do the recognition task, since the main tag associated to the image is usually the one linked to the object that takes the most space. For example, if we take a closer photo of a cat, we can expect the cat to get the white area around it while the rest of the picture is identified as background, but if we take a picture of the neighbourhood and an alley cat appears on it, it won't be that important, so the cat will be identified as part of the background. We can confirm this behaviour with the green stem in the right corner: it is probably big enough to preserve its shape through all the convnet, but is not the main focus on the picture.

Conclusion

We can conclude that the first block is very sensitive to small details and contains a lot of information, that is why we can recreate almost perfectly the content image. Meanwhile, block two is cappable of discern what is background and what not. In order to do that, it probably identifies the most used color (white in the first exaple, black in this one) and uses that information to 'classify' the remaining parts of the image as important, hence the white area. Like we said previously, the color extraction begins now, but is not as noticeable because the object is not only complex regarding to its shape, but also to its texture. In this case the yellow areas seem to fade a little while mantaining the general shape, and given the red color palette of the flower, it is easy to spot the green noise pixels. The green stem and the leafs have dark blue noise, probably derived from the background.

Block 3

As we saw before, the color extraction inside the objects intensifies. If we compare the layers of this block, the remaining black color around the corolla is removed, highlighting the silhouette of the flower and its petals. This process also enhances the contour of the petals, the drops and the pseudanthium or flower head: we can see that the petals acquire edges and thus the convnet will be able to recognize them as single entities. However, as we could observe in the previous example, this borders are not accurate: if we look at the petals of the right flower, they are bigger in the third layer than in the first layer (figures A.27, A.29 respectively).

The disk floret is also an interesting zone. In the first layer, the anthers have a lighter color (yellow) than the rest of the disk (dark brown, red, black). In the third layer, this colour palette changes to a lighter one for the anthers while the pixels from the disk turn into dark blue, purple... If we compare the second and third layer, this change makes the points stand out more at a glance. This turns that zones into blobs that will be easy to detect later after more convolutions.

Conclusion

To sum up, in this block the convnet starts to divide the big objects into smaller ones, capturing the little details that would be used later to classify the object. This is achieved by extracting the color inside of the objects, enhancing the edges. The convnet is also able to change the colour of certain parts of the image in order to avoid mixing artifacts and thus creating blobs that will be detected after several convolutions.

Block 4

In the first layer (fig. A.31) the contour of the flower changes again and becomes inaccurate. This is the same handrown effect that we mentioned before. We can see some blue-purple and green stains in the petals. In the second layer (fig. A.32) this stains get bigger and affect all the flower. This layer seems to be the last one to mamtain more or less accurate data about the shape. In the third layer (like the last block) structures tend to loose their borders because they merge with other borders (see fig. A.33). Some of the edges are completely unreal as a result of the reconstruction progress.

Regarding to the blobs in the disk floret, we can see that our theory was correct: the white dots in the disk floret are still clearly visible and have preserved the white color. They help to distinguish the disk from the corolla of ray floret along with the green phosphorite and purple areas. Again, if we compare layers two and three, it seems that the convnet is applying a gaussian filter that blurs the image. As we noted before, the small objects, like the stalks, are missing, blended with the background.

Conclusion

In this block, we start to lose significant information about the edges of the individual parts of the objects. This causes the dissapearance of little objects, leaving only the most significant ones.

Block 5

In this block the first two layers still keep some structures that resemble the petals and the disk floret. However, there are a lot of edges that did not exist in the original image, creating the illusion of even more petals. The border of the petals has now a purple blob that can be used to identify them in the third layer, the one with the least information. The disks of both flowers (along with the corolla) are the only relevant parts at this level: the green section in the right corner, the leaves and the stems have dissapeared completely.

4.3 Content mixing between two images

In this section we explore the possibility of mixing content from two different images. In order to do this, we are going to expand and combine the previous examples: we are going to change a random image with gradient descent to minimize the content loss corresponding to both sample images at the same time.

Our results show that content mixing on the first blocks generates an image that is merely an overlap of both samples, depending on the weight assigned to each one. On deeper levels, the content from both images mixes as we would expect, since in those levels the content is more flexible. However, because of the lack of information inherent to those blocks, the output is not so pleasing, and the results will need extra processing to optimize their appearance.

One of the pictures is a mugshot of the actor Chace Crowford, arrested by mistake on July 4, 2018 (fig. A.40). The other one is a mugshot of Terry Bailey, a 22 year old man, convicted felon, arrested for felony weapon charges on June 18, 2014 (fig. A.39).

The first attempt of content mixing that we tried was performed in the second layer of the second block. As we know, this layer still has almost all the content information, but the style and colour begin to change noticeably. Initially, we would expect an amalgam as the result, a picture where both heads are fused and the facial features are blended. Nevertheless, the results obtained are not as expected.

If we look at the pictures (fig. A.41, A.42 and A.43), when both sample images are given the same weight we obtain an image that contains both samples overlapping. Both faces have their transparency affected, presumably to fifty percent, allowing us to see at the same time both images. If we change this percentages, the transparency changes too: the heaviest one will be more solid and it will be more difficult to see the other one. We can not appreciate any details from one image if the weight exceeds the ninety percent. We can notice some subtle changes in the colouring of the image: the hair of Crowford displays a lighter brown color, probably affected by Bailey's skin color.

When we apply this method in block four or five, the results are reversed. We do obtain a picture where the facial features are blended: if we look at the results, the left eyes and ears are mixed, and Crowford's lips are distorted to match Bailey's. The right part of the faces is more difficult to combine as the features are not spatially close enough, but some of them, like the jawlines and necks, appear to be fused and thus creating a single bigger feature. Unfortunately, while this blocks are lax enough to allow this content mixing, the visual appearance of the results is not as satisfying, because the original color is lost due to how little information this layers have (see fig. A.44, A.45 and A.46).

If we try to mix the contents of layers in different blocks, this does not work (fig. A.47). The reason is that every block contains twice as many pixels, and thus information, as the previous block (except the last one). For this reason, if we try to mix something like the second layer of the second block with the third layer from the third block, the content of the second block will overshadow the content from the fourth one. In the example, we obtain the same result that we would obtain applying the method of the previous section to the image in the second block. The contents of the second image, the one in the fourth block, can not be distinguished.

Conclusion

This experiment tried to achieve similar results to the morphing technique. Regarding to the first part of the experiment, the results were negative, because what we get is just an overlap of the images while keeping the original color and appearance of the original samples. Conversely, in the second part of the experiment (working with the deepest layers), we obtain a picture where the facial features are blended. However, we do not have enough style information in those blocks to generate a visually pleasing output. Finally, if we try to mix content from different layers, the information of the deeper layer is not distinguishable.

4.4 Style transfer between two images

We have finally arrived to the main focus of this very project. In this section, we are going to apply the method explained in the third chapter to extract the style from a sample image and the content from another one, and then apply them to a random image to obtain a mix between both. In this section, we are going to discuss the results of some experiments regarding to the style transfer, quality of images and so on.

4.4.1 Experiments carried out on the parameters.

Number of iterations

One of the problems that we observed while doing the single-feature example was that every iteration of the gradiend descent method produced very little changes on the image. This means that we had to do a lot of iterations to obtain a significant output with observable results at a glance. We also tried to incorporate another feature related to color such as histogram manipulation, but the method tend to get stuck in local minimums.

For this reason, we decided to change to another version of the algorithm, and we chose L-BFGS. In terms of speed, each iteration of the algorithm would need an average time of twenty five minutes to complete. This experiments were performed in an old computer with no GPU, which would have speed up the process. As we can see in the pictures, there is little to no difference between doing five iterations to ten iterations (compare fig. A.49, A.51), because the algorithm converges really fast. The only noticeable difference is in the background where some of the brick lines start to dissapear.

Depth of layer in content

We wanted to see the differences between the Gatys et al (2015) approach and the supposedly improvement of Johnson et al (2016). In the Gatys version, we calculate the content feature from the second layer of the fourth block, while Johnson uses the second layer from the second block.

If we take a look at the results, we can barely see the contour of the baby on the brick wall when using the Gatys version. It is possible to distinguish both hands, both eyes, ears and mouth (fig. A.56). We ran the experiment twice to see if this was coincidental, but we can see that the general vague structure remains there (fig. A.58). There are differences regarding the exact shape of the eyes or the background bricks, for example, but those were expected. We must take into account that we are using a very regular pattern as the style sample: if we use a less regular pattern as the background such as a pile of rocks, it becomes increasingly difficult to see the features of the baby (fig. A.57). This is because Gatys has chosen a layer where we still have enough information of the original structure, but the content loss is very noticeable.

If we use the Johnson technique, the results display a much detailed content: be can clearly see the baby's face and several little details like hand lines, hair texture, the internal part of the ears... This remains true even if we use a non regular pattern such as the pile of rocks. On the one hand, this approach seems more appropiate if we are dealing with faces or other structures that require a high level of detail to be recognized. On the other hand, because of this rigidity regarding to the content, we should consider a different approach if we are trying create a paraledoia like the paintings of Giuseppe Arcimboldo.

We have also tried other layers: the first block is very similar to the second one, as they are the ones that retain the most information, while in the fifth block is almost impossible to distinguish any shape remotely cole to the original sample. We have added some pictures created with the third block (fig. A.54, A.55). Interestingly enough, the remaining content looks very similar to the results from the second block, but the texture pattern is less regular (we can compare the background brick lines between the second and third block).

To sum up, Gatys approach should be used when the content does not include a lot of detail structure, and should be reserved for landscapes or simple big objects with very well defined shapes and edges, while Johnson is more suited to faces or pictures with many little objects.

Proportion between style and content

A large set of results have been obtained with the same proportion of style and content: 99% style, 1% content. This may be ok to illustrate the previous cases, because we were not looking for an specific visual effect. In this section, however, we want to show the results of changing this proportion in the visual output. We have chosen a picture of the actual Batman as content and a flock of bats as style.

The first thing that we need to notice is that size, in this case, matters: the calculation of the content and style loss (especially the latter) and the gradient descent get slower depending on the number of pixels of the image. On the one hand, we want to work with little images. On the other hand, the more bigger is an image, the more details it has, so using a bigger image results in better style transfers because the samples are richer and the destination has more room to allocate all the features. This is the reason why we used the size (512,512), because a picture with this dimensions is big enough show complex structure details and the image processing does not take so long.

In the results we can see what happens if we work with a smaller size such as (128,128): the picture is so small that we can not appreciate any structural details, only a vague silhouette (fig. A.67). This is another side effect of working with smaller sizes: since the image can not contain many details because of space constrains, the structures of the style features are semantically bigger, because we do not have enough room to separate them. For example, the pupil can not be separated from the eyelid. When they are transferred to the output image, the structure brings with it some structure information from the style sample, causing some content loses.

If we use a value close to the fifty percent, the output obtained is very similar to the original image, because the content structure keeps some of their original style (see fig. A.61). When we increase the weight of the style close to 80%, we can begin to appreciate the appearance of bats from the style image. However, due to resizing issues, the level of detail of this bats is not actually that high, so we just get a black stamped pattern design. If we keep increasing the style weight, the content starts to dissapear while being substituted by the content from the style sample.

We can also do a comparison on what happens if we reduce the total variation value: when this is set to a low value such as 0.1, we obtain an almost plain background, reducing the amount of residual blurred bat shapes. If change the value to 1.0, the appearance of this shapes increases drastically, creating a pattern in the background (fig. A.66). We can confirm that if this value is high, the smoothness of the image decreases, as mentioned in chapter three.

Problems derived from resizing

As we have mentioned previously, we decided to simply resize the images when they did not fit our criteria: they are too big or rectangular. If we had to deal with a very big images, the duration of each iteration may be increased to an unaceptable point, so we would need to divide it into square patches, process each one of them and then try to combine them to create the final output. Since a patch can be very different from another one, we could end up with a non homogeneous picture. We could try to do some sort of interpolation between the patches, but this is not trivial and requires an extra effort when applying the content or the style [22].

If the image is not too big, the problem of working with rectangular images is easy to solve: what we need to do is to reduce the image keeping the aspect ratio if necessary and then crop a rectangular section of it. The cropped section will be the sample for our algorithm. As we can see in the following results (fig A.80), when we use the cropped sections of the images, the results are more detailed and the style fits better. We can notice the facial hair and other features are very detailed, unlike the resized version (fig. A.73). Also, we need to highlight the fact that the output version has color while the original picture is a black and white image: the red and blue sections reproduce more faithfully the selected style than the resized version.

The last part of this experiment was to find out what happens if the style image is not big enough. What we tried here was to copy the image several times to keep the ratio while generating a bigger image. The problem with this approach is that we can create false features derived from the zone where two copies come together. If we look at the results, the expanded style reproduces better the contour, but some content details are worse: the pockets in Batman's belt are more fuzzy than the ones obtained with the simple style image, for example (fig. A.83). Regarding to the style, the non-expanded version has an almost plain background, while the expanded one presents some 'structural noise' caused by the resize process. We can not decide which one is better, as the answer depends on the viewer's taste.

Modifications to style extraction

In this experiment we tried to know what happens visually if we changed the layer list that we use to calculate the style loss. In the pictures we have a picture of a woman before and after using makeup (fig. A.84). The other two pictures show the result of applying the picture with makeup as style to the one without it. The idea behind this was to apply the makeup to the woman and obtain something similar to the picture with makeup.

As we can see in the pictures (fig. A.85), there is some difference in the results. In one picture we have used our typical set of layers: 'block1_conv2', 'block2_conv2', 'block3_conv3', 'block4_conv3', 'block5_conv3'. In the other one we have used only two: 'block1_conv2', 'block5_conv3'. Both images take the color from the blouse and and apply it to the T-shirt, changing its color from black to hickory brown. They also change a little the skin tone to a brighter one, but they are not able to reproduce the lipstick color nor the eyeliner and the mascara. This is because the structures associated to that zones are not present in the version without makeup. Since the images are so similar, these structures probably do not have enough room to be transfered to the other picture.

If we take a closer look, the 'fully styled' picture reflects better the style, because it takes the yellow tone from the hair and tries to add it to the global tone of the picture: notice the yellow stain in the background close to the hair and the dyeing effect on the roots. In the other picture, we can see a slight lightening effect on the hair (not as strong as in the fully styled one), the background color and the skin tone, which is similar to the one with makeup.

To sum up, the differences in the list of layers are noticeable, but if we are not looking for a specific visual output, it is difficult to spot the differences. Regarding to style transferring, we are not able to take a complex structure (like the eyeliner) from the style sample and combine it with the content one, no matter how similar both pictures are.

Final example of style transfer

We have included an example of how the style transfer replicates the painting technique from the style sample. If we look at the elephant picture (fig. A.88), we can see how the parts with grass or vegetation have a very similar texture like the grass from the style image. If we look closely to the sky, the texture used is plain, matching the flat area next to the girl in the right and a little zone in the upper left corner. The pattern in the elefant, the parts of the floor without grass and the house on the left are probably from the clothes of the kids. Overall, it is possible to match both content and style with the resulting image, which was our goal since the begining of this project.

5

Conclusions and future work

5.1 Conclusions

The aim of this project is to act as an introduction to working with neural networks and image transformation. In the second chapter we have tried to give some insights about the style transfer problem: its origin, evolution and actual state. In the third chapter we have learned about how convolutional networks work and about concepts commonly associated to them such as backpropagation or gradient descent. We have also learned how to apply all this work in a practical way with an specific example.

In terms of visual appearance, the neural style shows distortions reminiscent of a painting, even if we use two photographs as the input. For this reason, this technique is not suitable for applications that require high levels of accuracy, somehow limiting the opportunities to use it. This technique is best suited for artistic environments or special effects.

Our implementation of the general method is very flexible, allowing us to change various parameters in order to answer all the questions mentioned in the first chapter while getting several extra answers as we saw in chapter four. This work also provides a link between all the mathematical aspects like partial derivatives, gradient descent, optimization... with very visual results in a funny and casual way, since this topics are usually a little bit too abstract for the average student. For this reasons, we believe that this project has reached its goal.

5.2 Future work

Based in the comparison of our results and the ones provided by popular applications and sites such as Ostagram.me, Prisma, Deepart.io... it is clear that we could do a refinement in our parameters in order to obtain results closer to the ones from this apps. To do that, we could try to use a genetic algorithm under some stylistic constraints to find more aesthetic outputs. Another option to improve the results is to select a style image that is already similar to the content one in certain aspects instead of a random one: same looking background, textures, composition, light conditions, high contrast...

We believe that this project has practical applications beyond artistic motivation in certain scenarios. For example, we could try to use an additional network to learn an specific transformation and then apply it in real time, as we can see in figure 5.1. This way, we could enhance the surveillance cameras' recording in natural areas to replicate light conditions (assuming they do not need high accuracy). We could also use it in videogame design opening new style possibilities for game assets, or incorporate it in special effects for background movies. In terms of research, we could expand the project adding new elements like pre-processing to the images or adding a new dimmensions in our problem either in the samples (2-D to 3-D models, for example) or in the optimization.

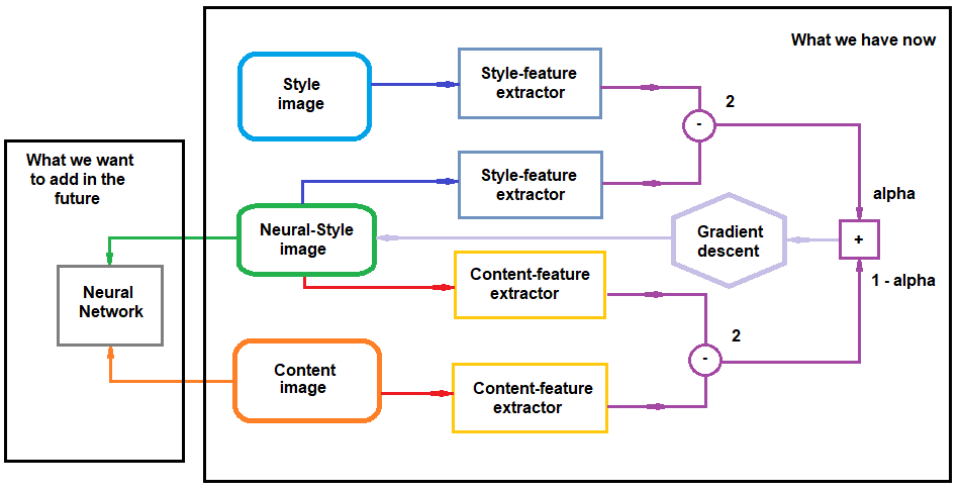


Figure 5.1: Using another neural network to learn a transformation and increase the performance

Glossary of acronyms

- **Blob**: A region of an image in which some properties are constant or approximately constant
- **Block**: In the VGGNet design, a group of layers that have the same size, along with the inmediate following pooling layer
- **Convnet** : Convolutional neural network
- **Gradient descent** : a first-order iterative optimization algorithm for finding the minimum of a function
- **IDE**: A software application that provides comprehensive facilities to computer programmers for software development
- **ILSVRC** : ImageNet Large Scale Visual Recognition Challenge
- **Kernel**: In image processing, a small matrix. It is used for blurring, sharpening, embossing, edge detection, and more. This is accomplished by doing a convolution between a kernel and an image.
- **L-BFGS**: A limited-memory version of BFGS, an iterative method for solving unconstrained nonlinear optimization problems that belongs to the quasi-Newton methods
- **Markov random field**: In the domain of physics and probability, this is a set of random variables having a Markov property described by an undirected graph.
- **Meme**: A meme is an idea, behavior, or style that spreads from person to person within a culture—often with the aim of conveying a particular phenomenon, theme, or meaning represented by the meme
- **Momentum**: In machine learning, a value between 0 and 1 that increases the size of the steps taken towards the minimum by trying to jump from a local minima
- **Morphing**: A special effect in motion pictures and animations that changes (or morphs) one image or shape into another through a seamless transition
- **Pareidolia**: Psychological phenomenon in which the mind responds to a stimulus, usually an image or a sound, by perceiving a familiar pattern where none exists
- **VGG**: Visual Geometry Group

Bibliography

[1] Mary Potter, Brad Wyble, Carl Hagmann, and Emily Sarah McCourt. Detecting meaning in rsvp at 13 ms per picture. 76:1 – 30, 12 2014.

[2] O. Hauk, M.H. Davis, M. Ford, F. Pulvermüller, and W.D. Marslen-Wilson. The time course of visual word recognition as revealed by linear regression analysis of erp data. *NeuroImage*, 30(4):1383 – 1400, 2006.

[3] John. Berger. *Ways of seeing*. Penguin on design. Penguin Books, 1972.

[4] David Marr. *Vision*, volume 1. W. H. Freeman and Company, 1 edition, 1982.

[5] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[6] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[7] Jerome Y Lettvin, Humberto R Maturana, Warren S McCulloch, and Walter H Pitts. What the frog's eye tells the frog's brain. *Proceedings of the IRE*, 47(11):1940–1951, 1959.

[8] Horace B Barlow. Summation and inhibition in the frog's retina. *The Journal of physiology*, 119(1):69–88, 1953.

[9] Lawrence G Roberts. *Machine perception of three-dimensional solids*. PhD thesis, Massachusetts Institute of Technology, 1963.

[10] Seymour A Papert. The summer vision project. 1966.

[11] Adolfo Guzman-Arenas. Computer recognition of three-dimensional objects in a visual scene. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE PROJECT MAC, 1968.

[12] Irwin Sobel and Gary Feldman. A 3x3 isotropic gradient operator for image processing. *a talk at the Stanford Artificial Project in*, pages 271–272, 1968.

[13] David E Rumelhart and James L McClelland. Parallel distributed processing, explotation in the microstructure of cognition-vol. 1: Foundations. *Computational Models of Cognition and Perception, Cambridge: MIT Press, 1987*, 1987.

[14] Tony Lindeberg and Bart M ter Haar Romeny. Linear scale-space i: Basic theory. In *Geometry-Driven Diffusion in Computer Vision*, pages 1–38. Springer, 1994.

[15] Matthew A Turk and Alex P Pentland. Face recognition using eigenfaces. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*, pages 586–591. IEEE, 1991.

[16] Randy Pausch, Jon Snoddy, Robert Taylor, Scott Watson, and Eric Haseltine. Disney's aladdin: first steps toward storytelling in virtual reality. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 193–203. ACM, 1996.

[17] Fletcher J. Buckley. Computer graphics proceedings, siggraph 96. ACM SIGGRAPH, August 1996.

[18] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[19] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015.

[20] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S. Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. *CoRR*, abs/1603.03417, 2016.

[21] Justin Johnson, Alexandre Alahi, and Fei-Fei Li. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016.

[22] Chuan Li and Michael Wand. Combining markov random fields and convolutional neural networks for image synthesis. *CoRR*, abs/1601.04589, 2016.

[23] Xueyuan Mei, Fabian Chan, and Tianchang He. Fast mixed style transfer. 2017.

[24] Fujun Luan, Sylvain Paris, Eli Shechtman, and Kavita Bala. Deep photo style transfer. *CoRR*, abs/1703.07511, 2017.

[25] Mark Grundland and Neil A Dodgson. Color histogram specification by histogram warping. In *Color Imaging X: Processing, Hardcopy, and Applications*, volume 5667, pages 610–622. International Society for Optics and Photonics, 2005.

[26] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[27] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks. *Google Research Blog. Retrieved June*, 20(14):5, 2015.

[28] Allen B Tucker. *Computer science handbook*. CRC press, 2004.

[29] Paul J Werbos. *Beyond regression*. 1974.

[30] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[31] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[32] Stephen Boyd. *Convex optimization*. Cambridge University Press, Cambridge, 2009.

[33] John D Head and Michael C Zerner. A broyden—fletcher—goldfarb—shanno optimization procedure for molecular geometries. *Chemical physics letters*, 122(3):264–270, 1985.

[34] Guillaume Berger and Roland Memisevic. Incorporating long-range consistency in cnn-based texture generation. *CoRR*, abs/1606.01286, 2016.

[35] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. *CoRR*, abs/1412.0035, 2014.

[36] Slav Ivanov. Picking an optimizer for style transfer, 2017.

A

Compendium of the generated images

In this annex we include all the pictures related to chapter four. The reason is that we have a lot of pictures and we could not include them in the main chapters without resizing them. This would remove some of the low-level details discussed in that chapter, making the images useless.

A.1 Analysis of a simple image

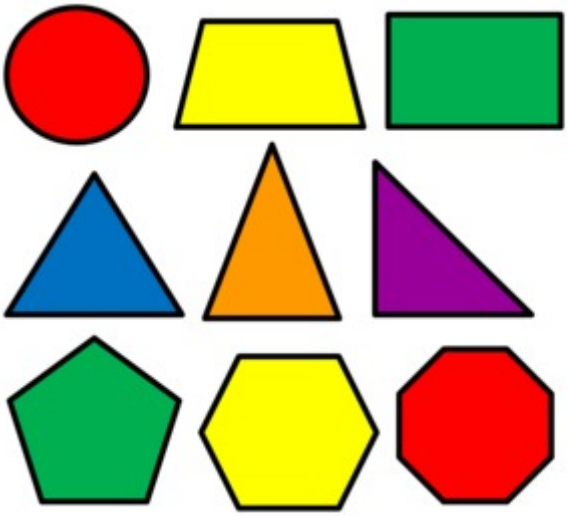


Figure A.1: Analysis of a simple figure: geometric figures

A.1.1 Block 1 and block 2

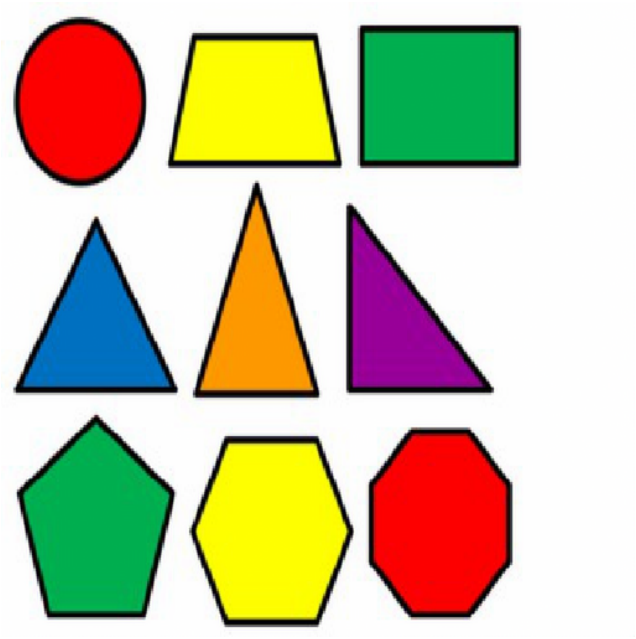


Figure A.2: Geometric figures: First layer of the first block

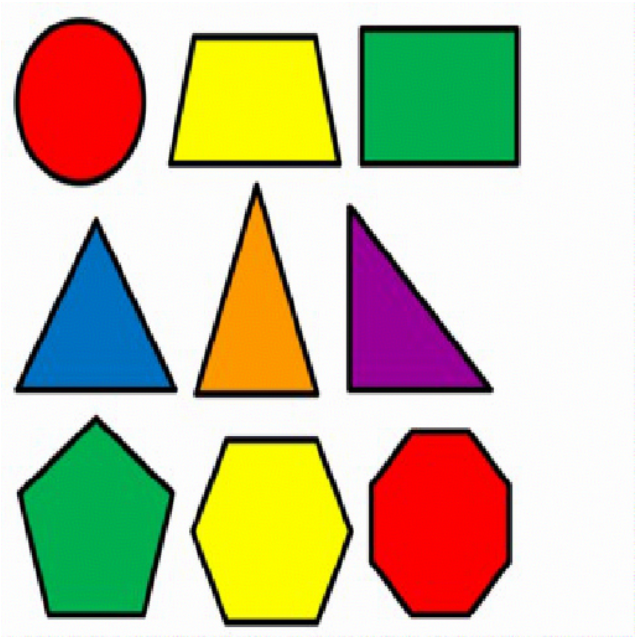


Figure A.3: Geometric figures: Second layer of first block

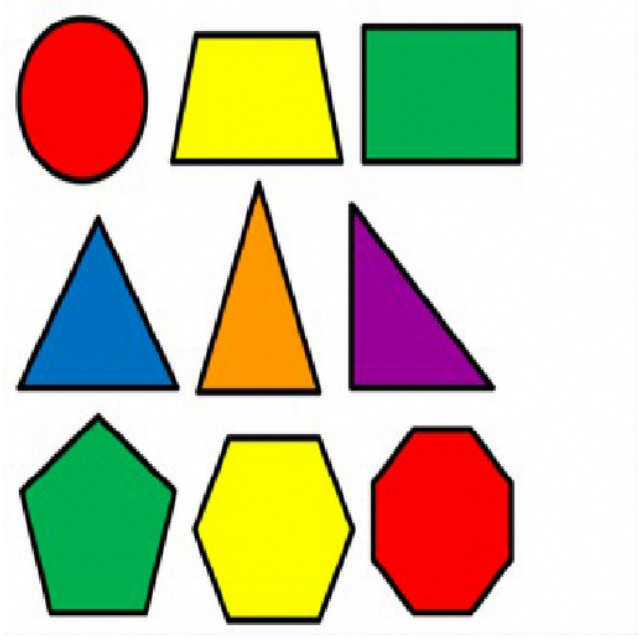


Figure A.4: Geometric figures: Pool layer of the first block

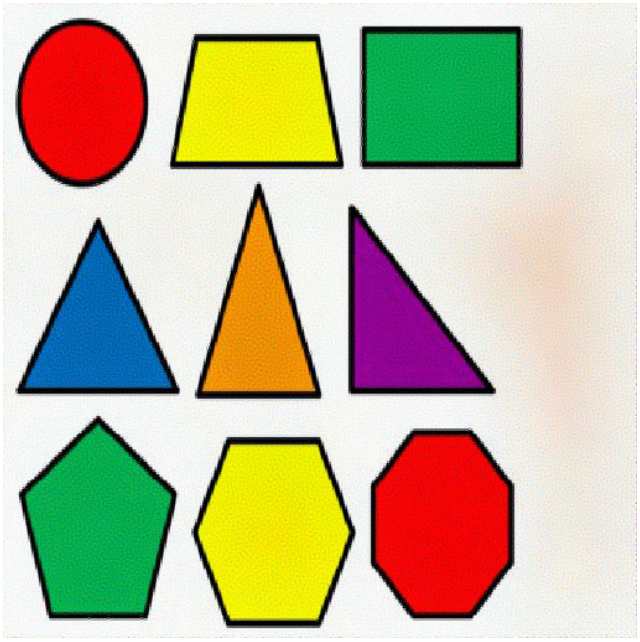


Figure A.5: Geometric figures: First layer of the second block

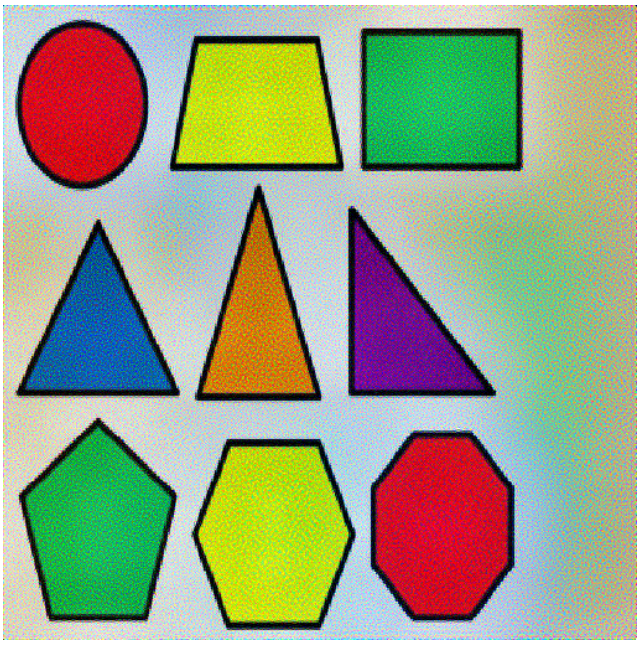


Figure A.6: Geometric figures: Second layer of the second block

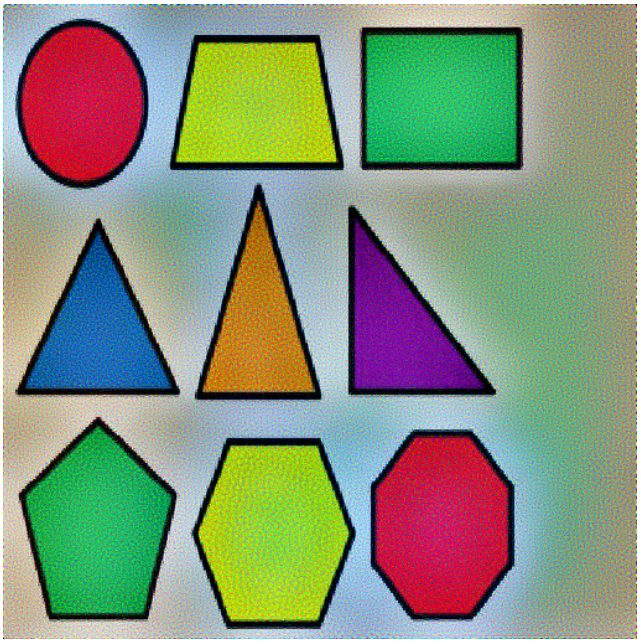


Figure A.7: Geometric figures: Pool layer of the second block

A.1.2 Block 3

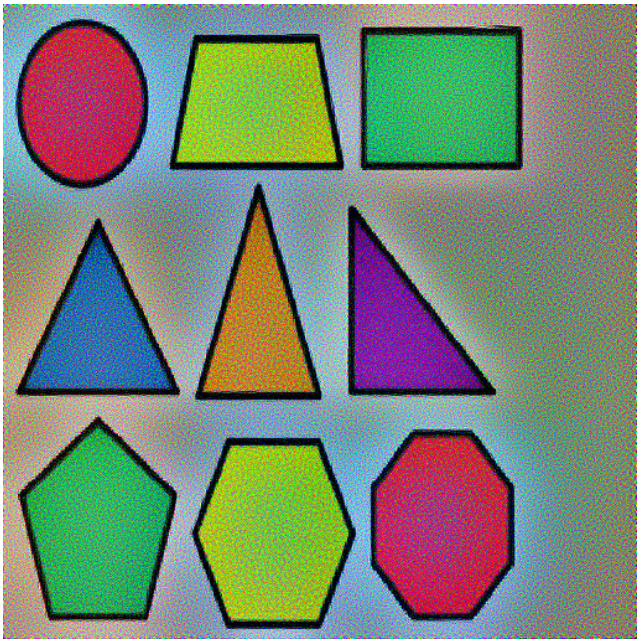


Figure A.8: Geometric figures: First layer of the third block

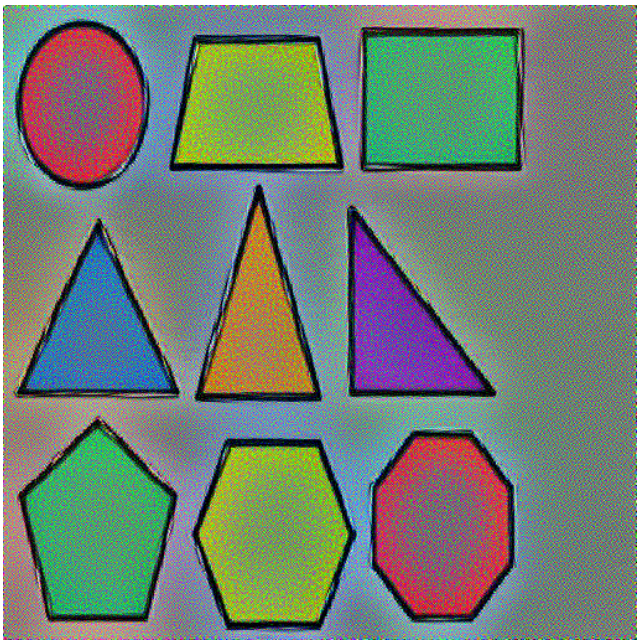


Figure A.9: Geometric figures: Second layer of the third block

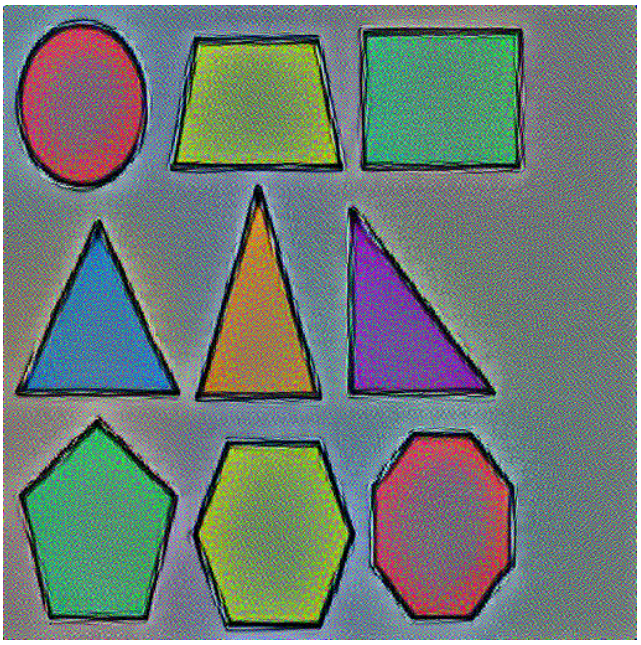


Figure A.10: Geometric figures: Third layer of the third block

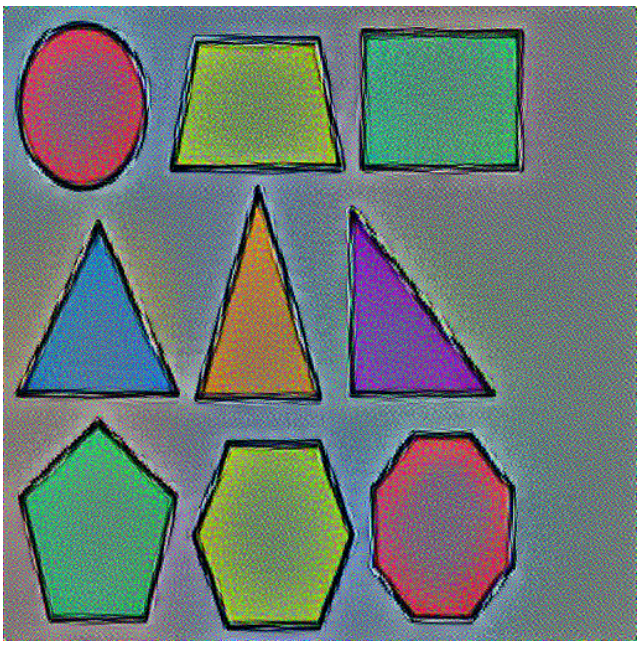


Figure A.11: Geometric figures: Pool layer of the third block

A.1.3 Block 4

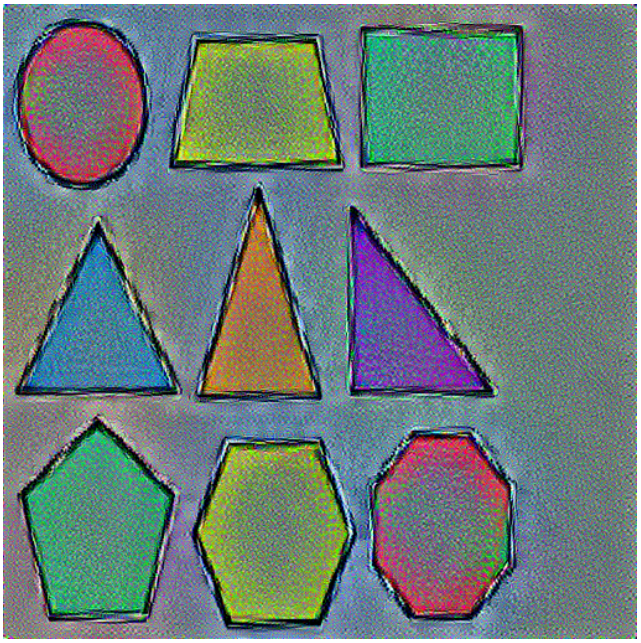


Figure A.12: Geometric figures: First layer of the fourth block

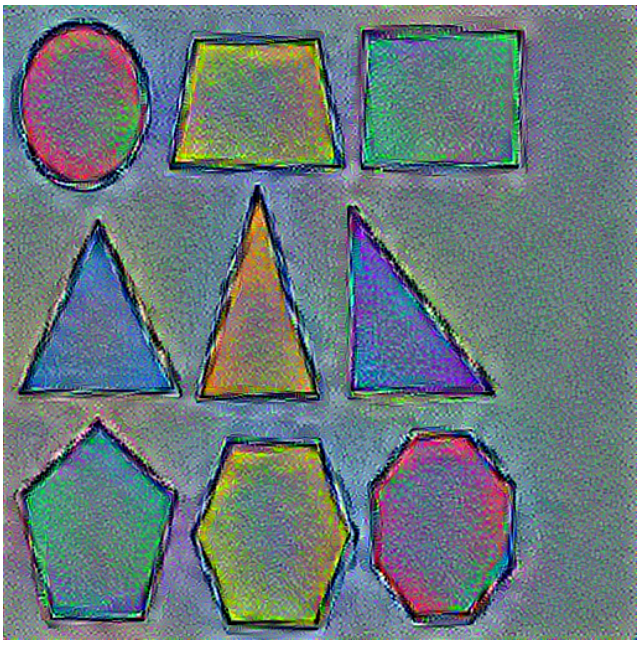


Figure A.13: Geometric figures: Second layer of the fourth block

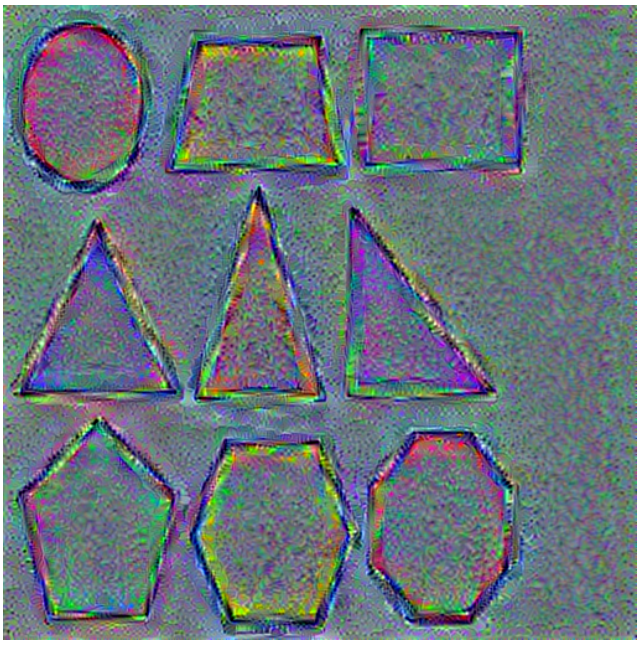


Figure A.14: Geometric figures: Third layer of the fourth block

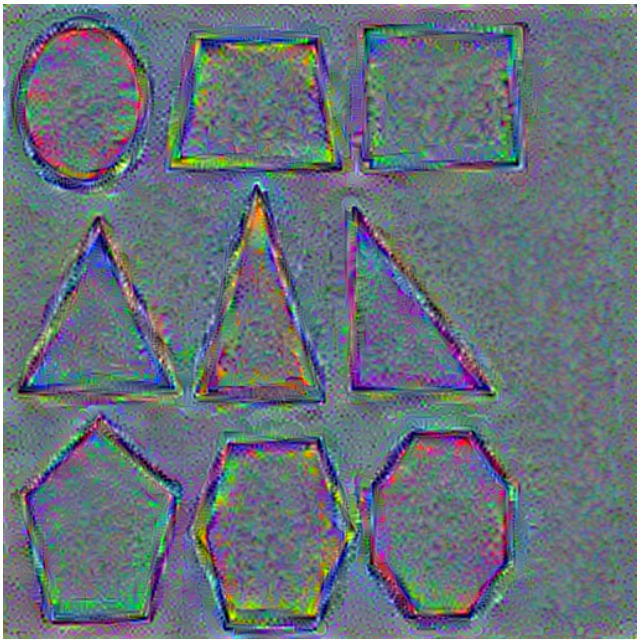


Figure A.15: Geometric figures: Pool layer of the fourth block

A.1.4 Block 5

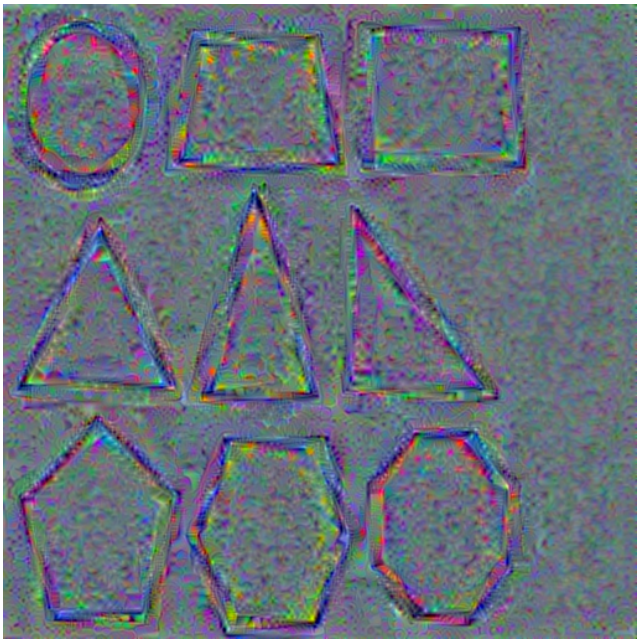


Figure A.16: Geometric figures: First layer of the fifth block

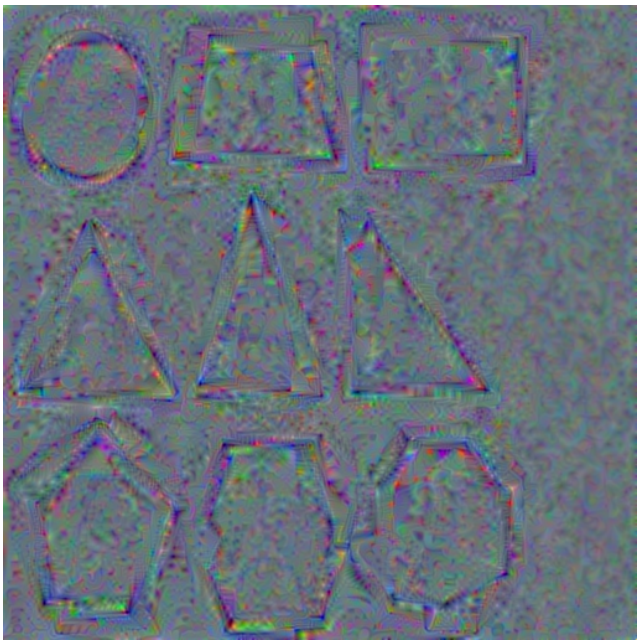


Figure A.17: Geometric figures: Second layer of the fifth block

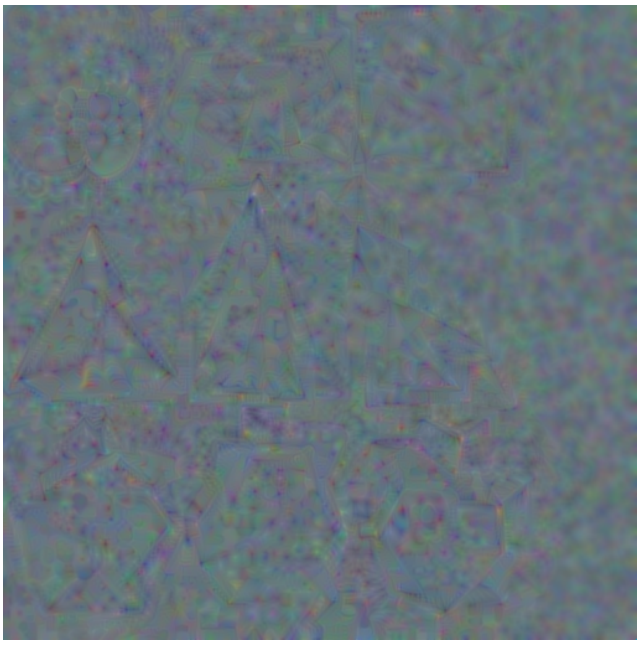


Figure A.18: Geometric figures: Third layer of the fifth block

Figure A.19: Geometric figures: Pool layer of the fifth block

A.2 Analysis of a complex image

Figure A.20: Analysis of a complex figure: Close-up of two flowers

The flowers in the picture are two Red Gerbera Daisy. Gerbera is a genus of plants in the Asteraceae (daisy family).

A.2.1 Block 1 and block 2

Figure A.21: Close-up of two flowers: First layer of the first block

Figure A.22: Close-up of two flowers: Second layer of the first block

Figure A.23: Close-up of two flowers: Pool layer of the first block

Figure A.24: Close-up of two flowers: First layer of the second block

Figure A.25: Close-up of two flowers: Second layer of the second block

Figure A.26: Close-up of two flowers: Third layer of the second block

A.2.2 Block 3

Figure A.27: Close-up of two flowers: First layer of the third block

Figure A.28: Close-up of two flowers: Second layer of the third block

Figure A.29: Close-up of two flowers: Third layer of the third block

Figure A.30: Close-up of two flowers: Pool layer of the third block

A.2.3 Block 4

Figure A.31: Close-up of two flowers: First layer of the fourth block

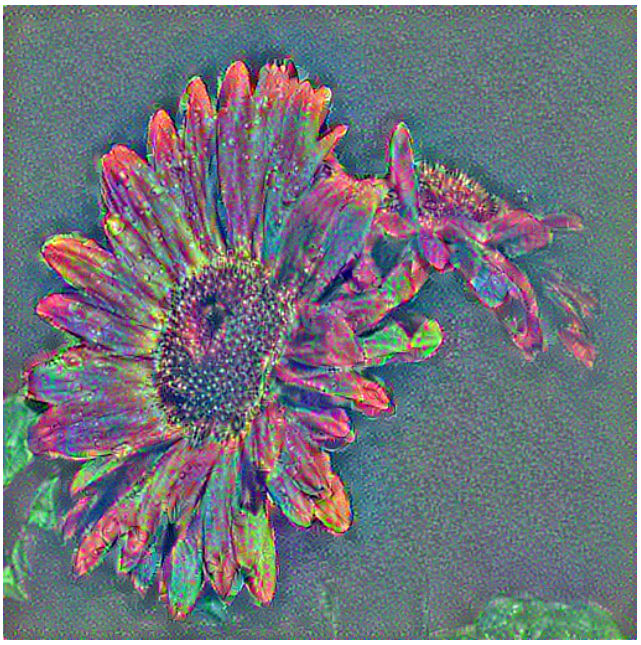


Figure A.32: Close-up of two flowers: Second layer of the fourth block

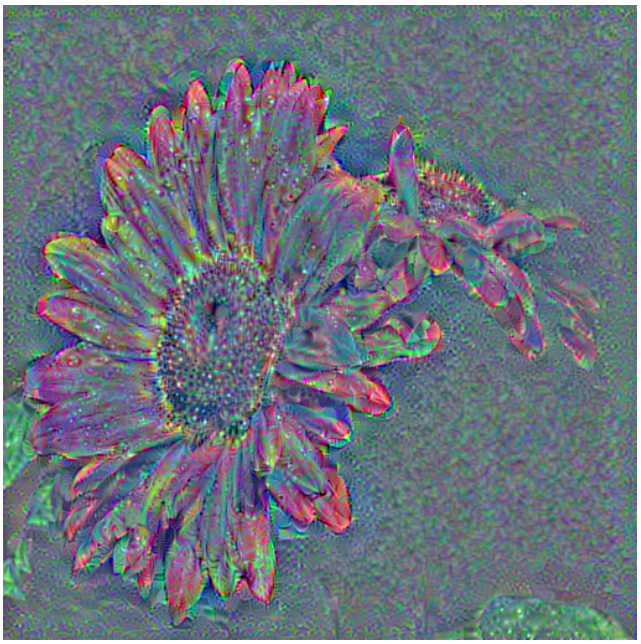


Figure A.33: Close-up of two flowers: Third layer of the fourth block

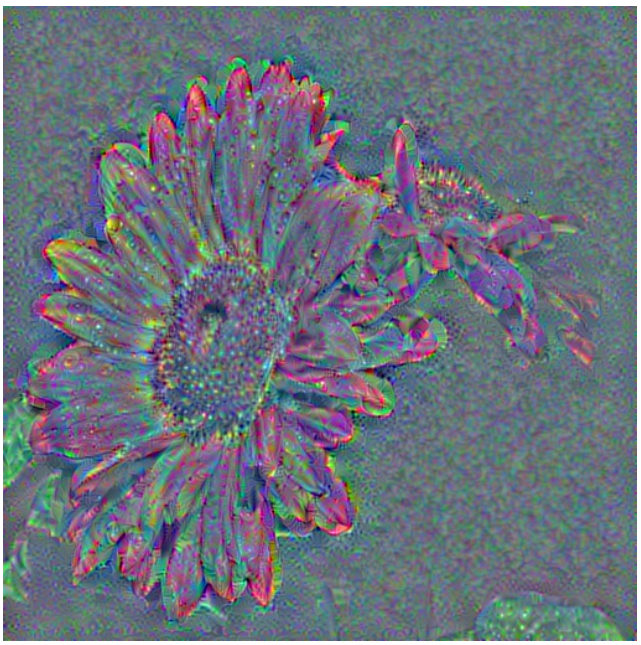


Figure A.34: Close-up of two flowers: Pool layer of the fourth block

A.2.4 Block 5

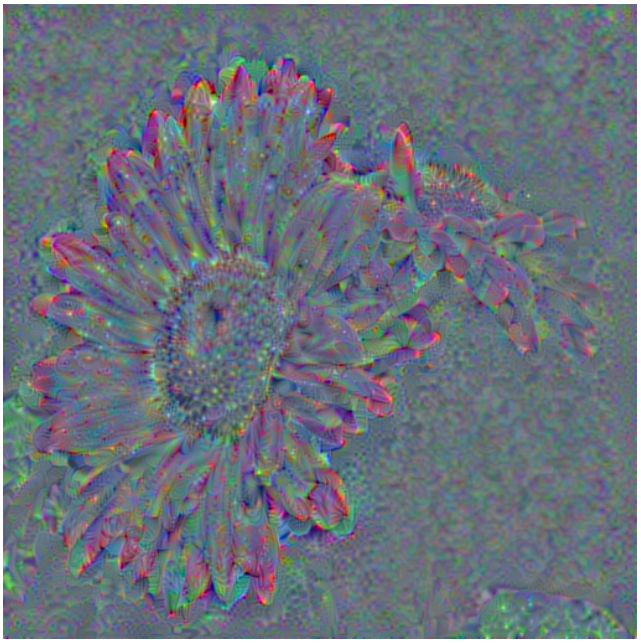


Figure A.35: Close-up of two flowers: First layer of the fifth block

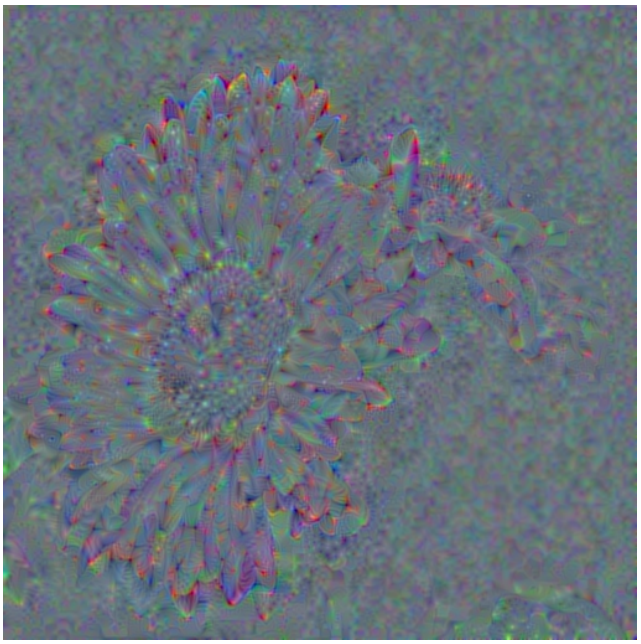


Figure A.36: Close-up of two flowers: Second layer of the fifth block

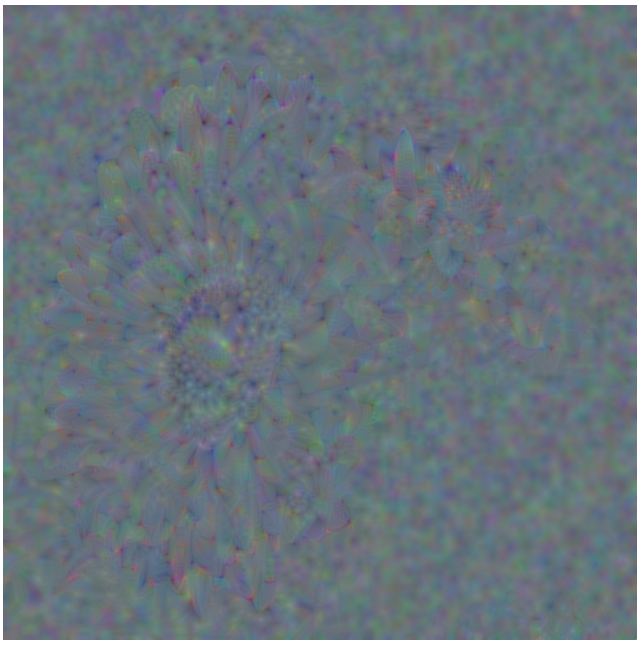


Figure A.37: Close-up of two flowers: Third layer of the fifth block

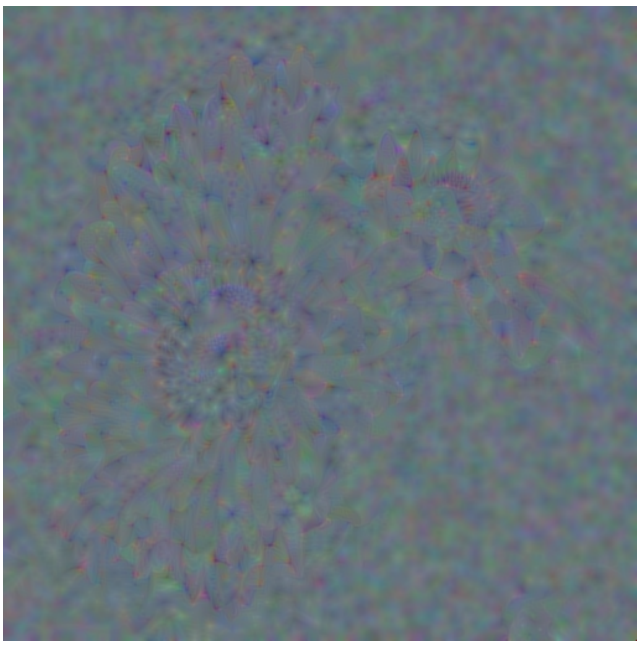


Figure A.38: Close-up of two flowers: Pool layer of the fifth block

A.3 Content mixing between two images

Figure A.39: Mugshot of convicted felon Terry Bailey

Figure A.40: Mugshot of actor Chace Crowford

A.3.1 Superficial layers

Figure A.41: Combination in superficial layers: 85% Crawford

Figure A.42: Picture combining both pictures with 50% each

Figure A.43: Combination in superficial layers: 85% Bailey

A.3.2 Deep layers

Figure A.44: Content combination in layer 3 of block 4

Figure A.45: Content combination in layer 1 of block 5

Figure A.46: Content combination in layer 3 of block 5, different content values but 50% ratio

A.3.3 Using layers from different blocks

Figure A.47: Mixing content from block 2 with block 4

A.4 Experiments carried out on the parameters.

Figure A.48: Picture of a baby smiling

A.4.1 Number of iterations

Figure A.49: Results after five iterations (brick wall style)

Figure A.50: Results after five iterations (rock style)

Figure A.51: Results after ten iterations (brick wall style)

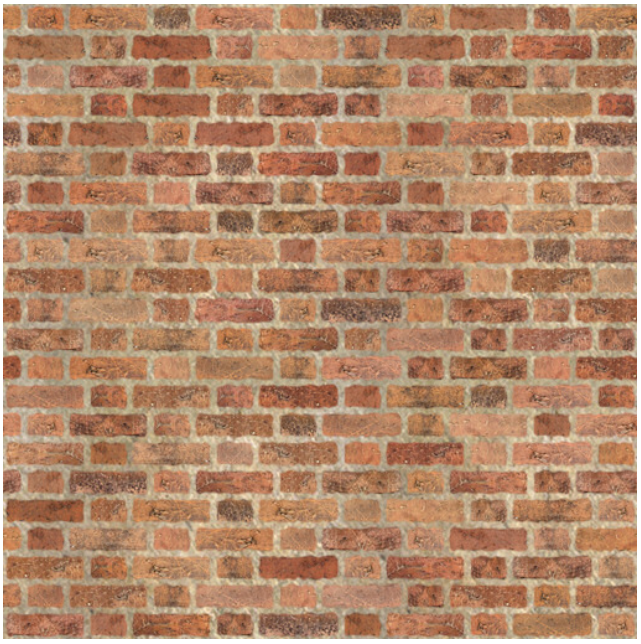


Figure A.52: Picture of a brick wall

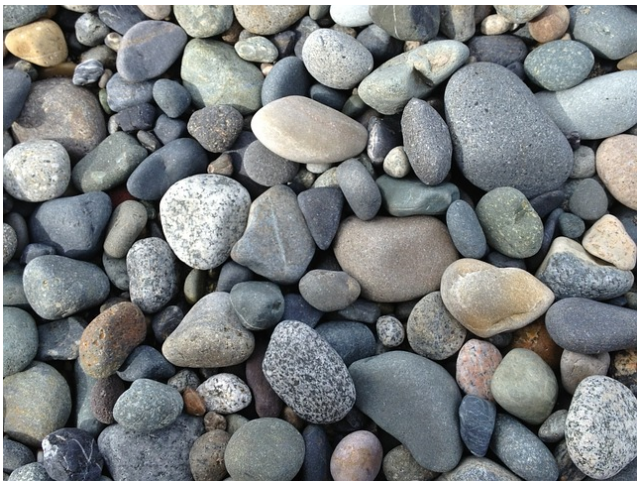


Figure A.53: Picture of a set of rocks

A.4.2 Depth of layer in content

Figure A.54: Results from the second layer of the third block (brick wall style)

Figure A.55: Results from the second layer of the third block (rock style)

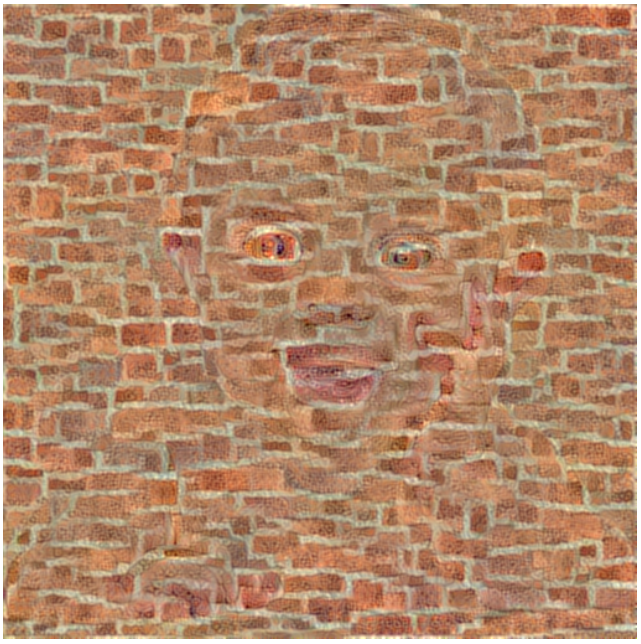


Figure A.56: Results from the second layer of the fourth block (brick wall style)

Figure A.57: Results from the second layer of the fourth block (rock style)

Figure A.58: Results from the second layer of the fourth block (brick wall style, verification)

A.4.3 Proportion between style and content

Figure A.59: Picture of Batman used as sample

This is a picture of Ben Affleck with his Batman's costume.

Figure A.60: First style sample: flock of bats

Figure A.61: Increasing the style value: 0.1

Figure A.62: Increasing the style value: 1

Figure A.63: Increasing the style value: 2

Figure A.64: Increasing the style value: 5. Total variation: 0.1

Figure A.65: Increasing the style value: 10

Figure A.66: Increasing the style value: 5. Total variation: 1

Results in 180x180 resolution

Figure A.67: Increasing the style value: 5. Total variation: 0.1. Size: 180x180

Figure A.68: Increasing the style value: 5. Total variation: 1. Size: 180x180

Changing the style image

Figure A.69: Second style sample: another flock of bats

Figure A.70: Increasing the style value: 5. Total variation: 0.1 (second flock)

A.4.4 Problems derived from resizing

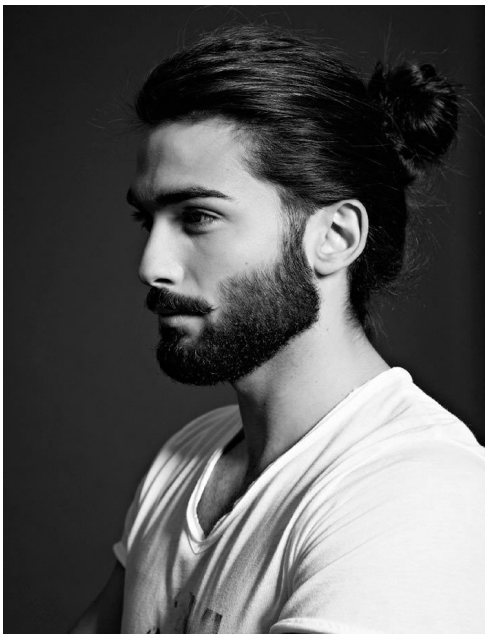


Figure A.71: Instagram picture: original size

Figure A.72: Daryl Feril illustration: original size

Figure A.73: Results: size 512x512. Style factor: 5

Figure A.74: Results: size 512x512. Style factor: 2

Figure A.75: Instagram picture: size 180x180

Figure A.76: Daryl Feril illustration: size 180x180

Figure A.77: Results: size 180x180
png

Figure A.78: Instagram picture: size 512x512 (cropped)

Figure A.79: Daryl Feril illustration: size 512x512 (cropped)

Figure A.80: Results from cropped images

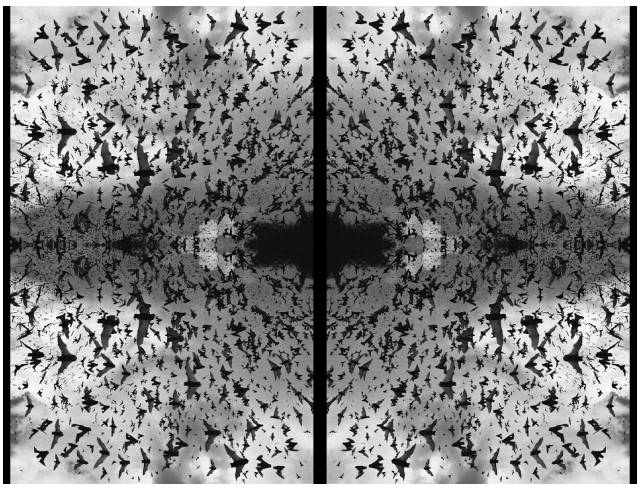


Figure A.81: Third style sample: first flock of bats (mirrored, extended)

Figure A.82: Testing mirroring effects in textures. Style value: 0.2

Figure A.83: Testing mirroring effects in textures. Style value: 1

A.4.5 Modifications to style extraction

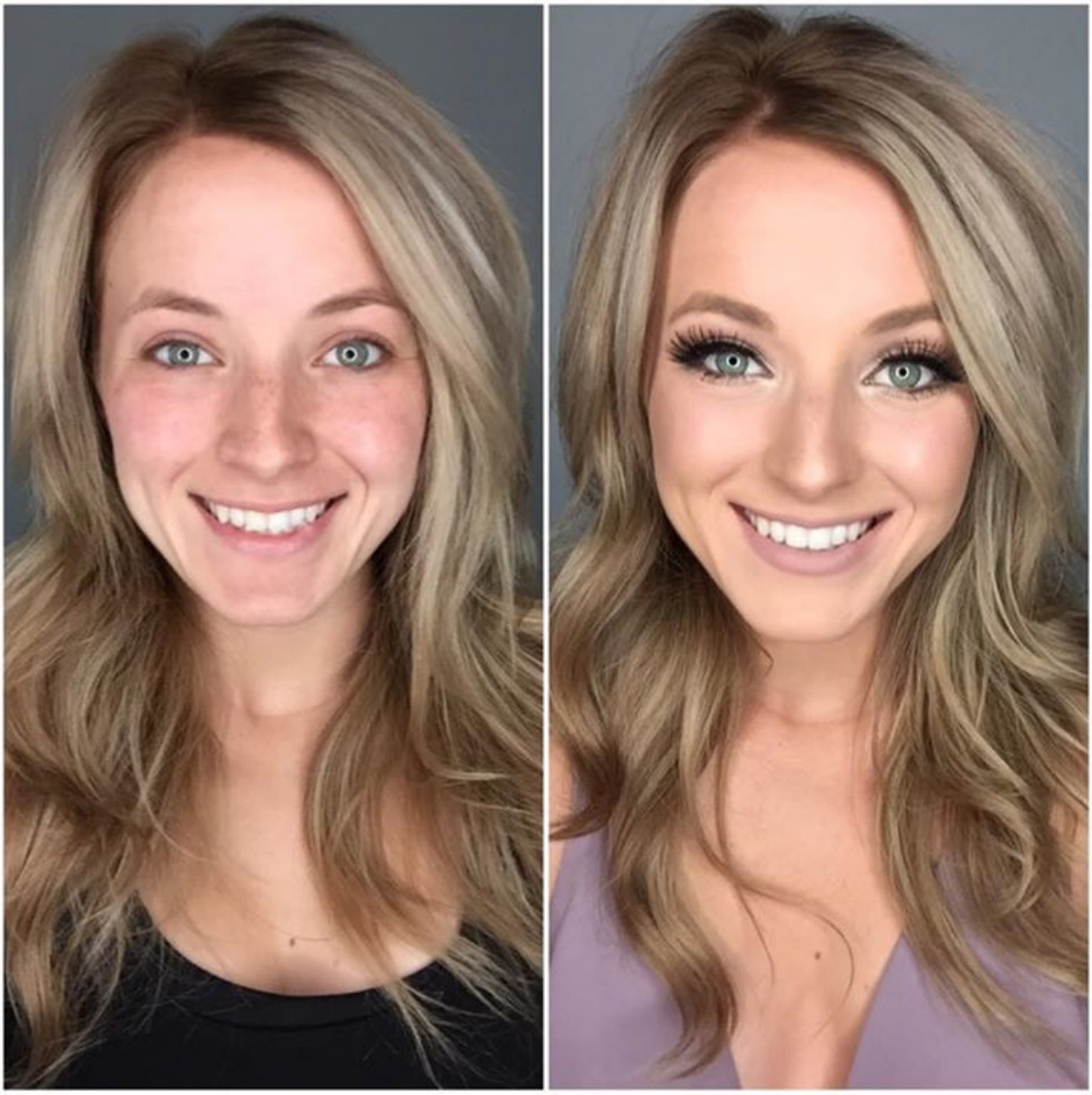
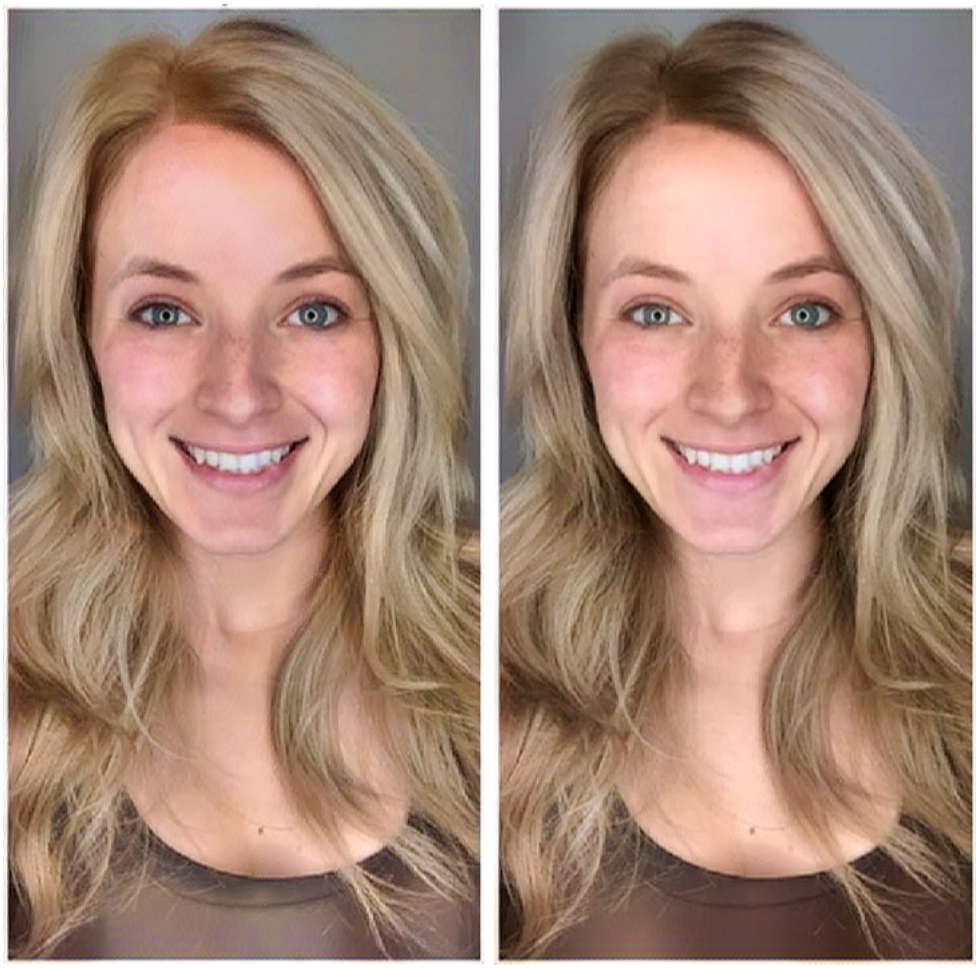


Figure A.84: Picture of a makeup advertisement (left: before, right: after)



(a) Using the standard set of layers (b) Using only two style layers

Figure A.85: Comparison between style effects changing the set of layers

A.4.6 Final example of style transfer

Figure A.86: Picture of an African bush elephant

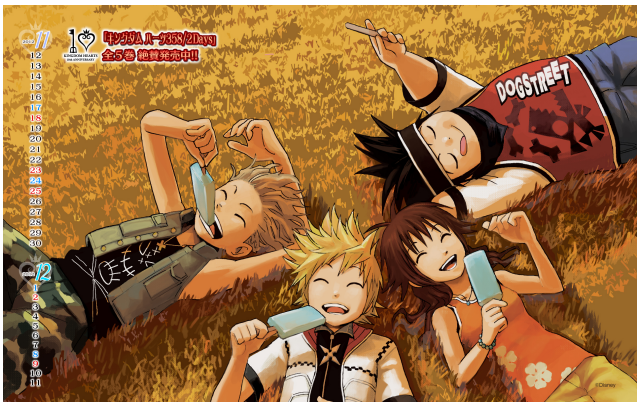


Figure A.87: Style sample: illustration of Shiro Amano for the anniversary of the Kingdom Hearts franchise

Figure A.88: Neural style example

B

Characteristics of the pictures

In this annex we will describe the parameters that were defined in order to generate all the pictures from the experiments. Ref is the reference of the image, H is the height, W is the width, C-I is the image used as content, S-I is the image used as style, C-W is the content weight, S-W is the style weight, TV is the total variation, C-L is the layer used in the content extraction and finally, S-L are the layers used in the style extraction.

Table B.1: List of parameters used

Ref	**H**	**W**	**C-I**	**S-I**	**C-W**	**S-W**	**TV**	**C-L**	**S-L**
A.2	512	512	A.1	-	5	5	1.5	block1-layer1	-
A.3	512	512	A.1	-	5	5	1.5	block1-layer2	-
A.4	512	512	A.1	-	5	5	1.5	block1-pool	-
A.5	512	512	A.1	-	5	5	1.5	block2-layer1	-
A.6	512	512	A.1	-	5	5	1.5	block2-layer2	-
A.7	512	512	A.1	-	5	5	1.5	block2-pool	-
A.8	512	512	A.1	-	5	5	1.5	block3-layer1	-
A.9	512	512	A.1	-	5	5	1.5	block3-layer2	-
A.10	512	512	A.1	-	5	5	1.5	block3-layer3	-
A.11	512	512	A.1	-	5	5	1.5	block3-pool	-
A.12	512	512	A.1	-	5	5	1.5	block4-layer1	-
A.13	512	512	A.1	-	5	5	1.5	block4-layer2	-
A.14	512	512	A.1	-	5	5	1.5	block4-layer3	-
A.15	512	512	A.1	-	5	5	1.5	block4-pool	-
A.16	512	512	A.1	-	5	5	1.5	block5-layer1	-
A.17	512	512	A.1	-	5	5	1.5	block5-layer2	-
A.18	512	512	A.1	-	5	5	1.5	block5-layer3	-
A.19	512	512	A.1	-	5	5	1.5	block5-pool	-
A.21	512	512	A.20	-	5	5	1.5	block1-layer1	-
A.22	512	512	A.20	-	5	5	1.5	block1-layer2	-

Note: the usual set of style layers (u) is [block1-layer2, block2-layer2, block3-layer3, block4-layer3, block5-layer3]

Table B.2: List of parameters used (cont)

Ref	H	W	C-I	S-I	C-W	S-W	TV	C-L	S-L
A.23	512	512	A.20	-	5	5	1.5	block1-pool	-
A.24	512	512	A.20	-	5	5	1.5	block2-layer1	-
A.25	512	512	A.20	-	5	5	1.5	block2-layer2	-
A.26	512	512	A.20	-	5	5	1.5	block2-pool	-
A.27	512	512	A.20	-	5	5	1.5	block3-layer1	-
A.28	512	512	A.20	-	5	5	1.5	block3-layer2	-
A.29	512	512	A.20	-	5	5	1.5	block3-layer3	-
A.30	512	512	A.20	-	5	5	1.5	block3-pool	-
A.31	512	512	A.20	-	5	5	1.5	block4-layer1	-
A.32	512	512	A.20	-	5	5	1.5	block4-layer2	-
A.33	512	512	A.20	-	5	5	1.5	block4-layer3	-
A.34	512	512	A.20	-	5	5	1.5	block4-pool	-
A.35	512	512	A.20	-	5	5	1.5	block5-layer1	-
A.36	512	512	A.20	-	5	5	1.5	block5-layer2	-
A.37	512	512	A.20	-	5	5	1.5	block5-layer3	-
A.38	512	512	A.20	-	5	5	1.5	block5-pool	-
A.41	512	512	A.39,A.40	-	0.005	10.0	2.0	block2-layer2	-
A.42	512	512	A.39,A.40	-	1	1.0	2.0	block2-layer2	-
A.43	512	512	A.39,A.40	-	5	0.025	2.0	block2-layer2	-
A.44	512	512	A.39,A.40	-	2	2	1.5	block4-layer3	-
A.45	512	512	A.39,A.40	-	5	5	1.5	block5-layer1	-
A.46	512	512	A.39,A.40	-	2	2	1.5	block5-layer3	-
A.47	512	512	A.39,A.40	-	2	2	1.5	blo2-lay2,blo4-lay3	-
A.49	512	512	A.48	A.52	0.025	5.0	1.0	block2-layer2	u
A.50	512	512	A.48	A.53	0.025	5.0	1.0	block2-layer2	u
A.51	512	512	A.48	A.52	0.025	5.0	1.0	block2-layer2	u
A.54	512	512	A.48	A.52	0.025	5.0	1.0	block3-layer2	u
A.55	512	512	A.48	A.53	0.025	5.0	1.0	block3-layer2	u
A.56	512	512	A.48	A.52	0.025	5.0	1.0	block4-layer2	u
A.57	512	512	A.48	A.53	0.025	5.0	1.0	block4-layer2	u
A.58	512	512	A.48	A.52	0.025	5.0	1.0	block4-layer2	u
A.61	512	512	A.59	A.60	0.025	0.1	0.1	block2-layer2	u
A.62	512	512	A.59	A.60	0.025	1.0	0.1	block2-layer2	u
A.63	512	512	A.59	A.60	0.025	2.0	0.1	block2-layer2	u
A.64	512	512	A.59	A.60	0.025	5.0	0.1	block2-layer2	u
A.65	512	512	A.59	A.60	0.025	10.0	0.1	block2-layer2	u
A.66	512	512	A.59	A.60	0.025	5.0	1.0	block2-layer2	u
A.67	180	180	A.59	A.60	0.025	5.0	0.1	block2-layer2	u
A.68	180	180	A.59	A.60	0.025	5.0	1.0	block2-layer2	u
A.70	512	512	A.59	A.69	0.025	5.0	0.1	block2-layer2	u
A.73	512	512	A.71	A.72	0.025	5.0	0.1	block2-layer2	u
A.74	512	512	A.71	A.72	0.025	2.0	0.1	block2-layer2	u
A.77	180	180	A.75	A.76	0.025	1.85	0.1	block2-layer2	u
A.80	512	512	A.78	A.79	0.025	2.0	0.1	block2-layer2	u
A.82	512	512	A.59	A.81	0.025	0.2	0.1	block2-layer2	u
A.83	512	512	A.59	A.81	0.025	1.0	0.1	block2-layer2	u
A.85a	512	512	A.84(a)	A.84(b)	0.025	5.0	1.0	block2-layer2	u
A.85b	512	512	A.84(a)	A.84(b)	0.025	5.0	1.0	block2-layer2	bl1-lay2, blo5-lay3
A.88	512	512	A.86	A.87	0.025	5.0	1.0	block2-layer2	u